Proceedings of

Grammar Systems Week 2004

Budapest, Hungary, July 5-9, 2004

Edited by

E. Csuhaj-Varjú and Gy. Vaszil

MTA SZTAKI, Budapest, 2004

Erzsébet Csuhaj-Varjú and György Vaszil (Eds.)

Grammar Systems Week 2004

Budapest, Hungary, July 5-9, 2004

Proceedings

Technical Report 2004/5
Theoretical Computer Science Research Group
MTA SZTAKI
Budapest

Foreword

Grammar Systems is a well-recognized field of formal language theory providing both syntactic models for describing multi-agent systems at the symbolic level and distributed models of language. The theory has been inspired and influenced by several scientific areas: 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 among the candidates for possible applications.

So far approximately four hundred publications have been published in the area, among other things a monograph in 1994, and a chapter of the Handbook of Formal Languages in 1997. Grammar systems was the topic of several Ph.D and M.Sc theses.

The workshop *Grammar Systems Week 2004* is the fifth one in a series of workshops started in 1995 in Mangalia, organized by Gheorghe Păun; the series continued in 1996, by a workshop organized by Erzsébet Csuhaj-Varjú in Budapest. The third workshop was organized by Alica Kelemenová in Brno in 1998, and the fourth one was organized by Rudolf Freund and Franziska Freund in Bad Ischl in 2000.

This volume is the proceedings of the *Grammar Systems Week 2004*, held in Budapest, July 5-9, 2004, under the auspices of the *European Molecular Computing Consortium* (EMCC) and the *IFIP Working Group 1.2 on Descriptional Complexity.* It contains the revised versions of papers presented during the event.

The members of the steering committee of the grammar systems workshops are Erzsébet Csuhaj-Varjú (Budapest, Hungary), Jürgen Dassow (Magdeburg, Germany), Rudolf Freund (Vienna, Austria), Jozef Kelemen (Opava, Czech Republic), Alica Kelemenová (Opava, Czech Republic), and Gheorghe Păun (Bucharest, Romania; Seville, Spain).

We would like to acknowledge the invaluable help of MTA SZTAKI, the Computer and Automation Research Institute of the Hungarian Academy of Sciences, in the frame of the project EU Centre of Excellence in Information Technology, Computer Science, and Control, contract no. ICA1-CT-2000-70025, HUN-TING, Workpackage 5. Last but not least, we thank the authors for participating and presenting their valuable work at the workshop.

Budapest, September of 2004

Erzsébet Csuhaj-Varjú, György Vaszil,
organizers and editors

Program

Monday, July 5

9.00 - 9.40	Erzsébet Csuhaj-Varjú (Budapest, Hungary): Opening. Grammar systems: Past, present, and future
9.40 - 10.30	Gheorghe Păun, invited speaker (Bucharest, Romania / Sevilla, Spain): Grammar systems vs. membrane computing
10.30 - 11.00	Break
11.00 - 11.40	G. N. Sathana Krishnan, Kamala Krithivasan, invited speaker, Ashish Choudhary (Madras, India): Distributed probabilistic finite automata
11.40 - 12.30	Jozef Kelemen (Opava, Czech Republic / Bratislava, Slovakia): Embodiment - A computational point of view
12.30 - 14.00	Lunch
14.00 -	Discussion session: General issues, descriptional complexity issues

Tuesday, July 6

9.00 - 9.50	Jürgen Dassow, invited speaker (Magdeburg, Germany): On cooperating distributed grammar systems with competence based start and stop conditions
9.50 - 10.40	Henning Bordihn (Potsdam, Germany), Markus Holzer (München, Germany): CD grammar systems as models of distributed problem solving, revisited
10.40 - 11.10	Break
11.10 - 11.40	Bettina Sunckel (Frankfurt am Main, Germany): On metalinear CD grammar systems
11.40 - 12.10	Suna Bensch, Henning Bordihn (Potsdam, Germany): Active symbols in pure systems
12.10 - 12.40	Liliana Cojocaru (Tarragona, Spain): On the time, space and communication complexity of cooperating distributed grammar systems
12.40 - 14.00	Lunch
14.00 -	Discussion session: Linguistic issues, descriptional complexity issues

Tuesday, July 6, continued

14.00 - 14.30	M. Dolores Jiménez López (Tarragona, Spain / Pisa, Italy): What can grammar systems do for linguistics?
14.30 - 14.50	Suna Bensch, Helmut Jürgensen (Potsdam, Germany): Modelling dialogues by grammar systems
19.00 -	Workshop Dinner

Wednesday, July 7

9.00 - 9.50	Alica Kelemenová, Michal Tupý (Opava, Czech Republic): Monocultures and homogeneous environment in eco-grammar systems
9.50 - 10.20	Peter Sebestyén, Petr Sosík (Opava, Czech Republic): Multiple robots in space: An adaptive eco-grammar model
10.20 - 10.50	Break
10.50 - 11.20	Francesco Bernardini, Marian Gheorghe (Sheffield, United Kingdom): Population P systems and grammar systems
11.20 - 11.50	Katalin Lázár (Budapest, Hungary): Eco-grammar systems: An approach to the crawlers' problem
11.50 - 12.20	Liliana Cojocaru (Tarragona, Spain): Parallel communicating pushdown transducer systems
12.20 - 14.00	Lunch
14.00 -	Discussion session: Evolutionary models, eco-grammar systems

Thursday, July 8

9.00 - 9.50	Rudolf Freund, Marion Oswald (Vienna, Austria): Modelling grammar systems by tissue P systems
9.50 - 10.30	A. Arun Prasath, Kamala Krithivasan, invited speaker (Madras, India): Distributed 2-way finite state quantum automata
10.30 - 11.00	Break
11.00 - 11.40	L. Kari (London, Canada), S. Konstantinidis (Halifax, Canada), E. Losseva, P. Sosík, G. Thierrin (London, Canada): DNA involutions and hairpin structures
11.40 - 12.20	Gemma Bel Enguix (Tarragona, Spain / Milan, Italy), M. Dolores Jiménez López (Tarragona, Spain / Pisa, Italy): Explaining language change with membranes
12.20 - 14.00	Lunch
14.00 -	Discussion session: Biocomputing, unconventional models

Friday, July 9

9.00 - 9.40	Henning Bordihn (Potsdam, Germany), György Vaszil (Budapest, Hungary): CD grammar systems with LL(k) conditions
9.40 - 10.10	K. S. Dersanambika, K. Krithivasan, K. G. Subramanian (Madras, India): Simple splicing grammar systems
10.10 - 10.40	Maria Adela Grando, Victor Mitrana (Tarragona, Spain): Can PC grammar systems benefit from concurrent programming?
10.40 - 11.10	Break
11.10 - 11.40	A. Roslin Sagaya Mary (Tarragona, Spain), K. G. Subramanian (Madras, India): Image splicing grammar systems
11.40 - 12.10	Results obtained during the workshop. Closing

Contents

Foreword **5**

Contributions **13**

A. Arun Prasath, Kamala Krithivasan: Distributed 2-way finite state quantum automata . . . 15
Gemma Bel Enguix, M. Dolores Jiménez López: A membrane systems explanation of semantic change . . . 30
Suna Bensch, Henning Bordihn: Active symbols in pure systems . . . 50
Francesco Bernardini, Marian Gheorghe: Population P systems and grammar systems . . . 66
Henning Bordihn, Markus Holzer: CD grammar systems as models of distributed problem solving, revisited . . . 78
Henning Bordihn, György Vaszil: CD grammar systems with LL(k) conditions . . . 95
Liliana Cojocaru: On the time, space and communication complexity of cooperating distributed grammar systems . . . 113
Liliana Cojocaru, Carlos Martin-Vide: Parallel communicating pushdown transducer systems . . . 126
Erzsébet Csuhaj-Varjú: Grammar systems: A short survey . . . 141
Jürgen Dassow: On cooperating distributed grammar systems with competence based start and stop conditions . . . 158
K. S. Dersanambika, K. Krithivasan, K. G. Subramanian: Simple splicing grammar systems . . . 170
Rudolf Freund, Marion Oswald: Modelling grammar systems by tissue P systems working in the sequential mode . . . 179
Maria Adela Grando, Victor Mitrana: A possible connection between two theories: Grammar systems and concurrent programming . . . 200
L. Kari, S. Konstantinidis, E. Losseva, P. Sosik, G. Thierrin: DNA involutions and hairpin structures . . . 212
Jozef Kelemen: On computational power of interacting bodies . . . 225
Alica Kelemenová, Michal Tupý: Monocultures and homogeneous environment in eco-grammar systems . . . 240
Gheorghe Păun: Grammar systems vs. membrane computing: A preliminary approach . . . 255
A. Roslin Sagaya Mary, K. G. Subramanian, K. S. Dersanambika: Image splicing grammar systems . . . 276
G. N. Sathana Krishnan, Kamala Krithivasan, Ashish Choudhary: Distributed probabilistic finite automata . . . 287
Peter Sebestyén, Petr Sosik: Multiple robots in space: An adaptive eco-grammar model . . . 300
Bettina Sunckel: On metalinear CD grammar systems . . . 313

Author Index **329**

Contributions

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 15 - 29, 2004.

Distributed 2-Way Finite State Quantum Automata

A. Arun Prasath, Kamala Krithivasan

Dept of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai, India -600036
kamala@iitm.ernet.in

Abstract

We define Multiple choice two-way quantum automata with multiple observables. Distributed quantum automata are defined with four modes of cooperation. We show that multiple choice two way quantum automata and distributed quantum automata have the same power as that of two way quantum automata(having single choice) with multiple observables.

1 Introduction

The possibility that a quantum computational model can be more powerful than its classical counterpart, as has been elucidated by the celebrated Peter Shor's factoring and discrete logarithm algorithms [2] and Grover's quantum searching algorithm [3] has encouraged researchers to come up with various quantum computational models. Two-way quantum finite state automata (2qfa) have been proposed as the quantum analogue of deterministic (2dfa) finite state automata, and it has been shown that these 2qfa's are strictly more powerful than the 2dfa's, the 2nfa's and the 2pfa's [1].

Also, in the past, there have been attempts to come up with co-operative distributed models of classical automata. In the case of push down stack automata, it has been shown that, distribution results in increased power [4]. In fact, it has been shown that the distributed push down stack automata are equivalent in power to that of a Turing machine and hence are computationally complete. The main aim of this paper is to analyze the effect of distribution on the power of computation in quantum automata.

The 2qfa model proposed by Kondacs and Watrous has one minor restriction. It is that only measurements with respect to one particular decomposition (observable) of the state space is allowed (that which decides if the machine halts with acceptance, halts with rejection or goes on). In this work, the 2qfa model is first generalized to a model ((2qfa-MOb)) that allows measurements with respect to one or several of multiple decompositions (observables) of the state space. The significance of the new model is understood when distribution of computation in quantum automata is attempted. Different modes of cooperation are defined similar to that in distributed finite state automata. Then, it is shown that the distributed quantum finite state automata (D-2qfa-MOb) can be reduced to a 2qfa-MOb, for any of its

modes of acceptance.

First, a multiple choice 2qfa-MOb (MC-2qfa-MOb) is formulated, and it is proved that the behaviour of a multiple choice 2qfa-MOb can be simulated by a 2qfa-MOb to an arbitrarily close accuracy. Then, the D-2qfa-MOb is formulated, and by reducing this to a MC-2qfa-MOb, in each one of its modes of cooperation, it is proved that a D-2qfa-MOb in any mode of cooperation, has the same power as that of a 2qfa-MOb, thus proving the result.

2 2-Way Quantum Finite State Automata (2qfa)

2.1 Definition

Kondacs and Watrous [1] define a 2qfa as a 6-tuple $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$, where Q is a finite set of states, Σ is a finite alphabet (The tape alphabet Γ is defined as $\Sigma \cup \{\#, \$\}$, where $\#, \$ \notin \Sigma$ are used to mark the left and right ends of the tape respectively), δ is the transition function for the automaton defined below, $q_0 \in Q$ is the initial state and $Q_{acc} \in Q$ and $Q_{rej} \in Q$ are the sets of accepting states and rejecting states respectively (Elements of Q_{acc} and Q_{rej} are halting states and elements of $Q_{non} = Q - (Q_{acc} \cup Q_{rej})$ are non-halting states. Also, $q_0 \in Q_{non}$ and $Q_{acc} \cap Q_{rej} = \emptyset$).

The contents of any tape can be described by a mapping $x : \mathcal{Z}_n \to \Gamma$, n being the number of distinct tape squares on the tape. The number of configurations of the 2qfa M on any tape x of length n is $n|Q|$, since there are n possible locations for the tape head and $|Q|$ internal states. The configuration of a tape can hence be described by a mapping $\mathcal{C}_n = Q \text{ X } \mathcal{Z}_n$. A superposition of M on a tape x of length n is any norm 1 element of the finite dimensional Hilbert space $\mathcal{H}_n = l_2(\mathcal{C}_n)$, which is the space of mappings from $\mathcal{C}_n$ to $\mathbb{C}$ with the usual inner product. Following Dirac's notation, for each $c \in \mathcal{C}_n$, $|c\rangle$ denotes the unit vector which takes value 1 at c and 0 elsewhere. All other elements of $\mathcal{H}_n$ may be expressed as linear combinations of these basic vectors. For a superposition $|\psi\rangle = \sum_{c \in \mathcal{C}_n} \alpha_c |c\rangle$, α_c is the probability amplitude associated with c in superposition $|\psi\rangle$.

The transition function δ of the 2qfa M is a mapping of the form, $\delta : Q \text{ X } \Sigma \text{ X } Q \text{ X } \{-1, 0, 1\} \to \mathbb{C}$. For each $q, q' \in Q$, $\sigma \in \Sigma$ and $d \in \{-1, 0, 1\}$, $\delta(q, \sigma, q', d)$ represents the amplitude with which a machine currently in state q and scanning symbol σ will change its state to q' and move its tape head in direction d. For any tape x, δ induces a time evolution operator U_δ^x on $\mathcal{H}_{|x|}$ as follows. $U_\delta^x |q, k\rangle = \sum_{q', d} \delta(q, x(k), q', d) |q', k + d(mod|x|)\rangle$ for each $(q, k) \in C_{|x|}$, and is extended to all of $\mathcal{H}_x$ by linearity.

Since valid superpositions for the automata are of unit norm, the finite dimensionality of $\mathcal{H}_n$ requires U_δ^x to be a unitary operator, so that any valid superposition will evolve into another valid superposition and the automata M is then well formed. Now, let $V_\sigma : l_2(Q) \to l_2(Q)$ be an unitary operator in the Hilbert space $l_2(Q)$, and let $D : Q \to \{-1, 0, 1\}$. Now, if the transition function δ is defined as

$$\delta(q, \sigma, q', d) = \begin{cases} \langle q' | V_\sigma | q \rangle & D(q') = d \\ 0 & D(q') \neq d \end{cases} \longrightarrow \text{Eqn.1}$$

M is well-formed iff V_σ is unitary.

2.2 Observables and Measurements

2.2.1 Definition

An observable $\mathcal{O}$ is a decomposition of the Hilbert space $\mathcal{H}_n$ into subspaces: $\mathcal{H}_n = E_1 \oplus \cdots \oplus E_k$, where the E_i are pairwise orthogonal. Let $|\psi_i\rangle$ be the projection of $|\psi\rangle$ onto E_i, for each i, so that, $|\psi\rangle = |\psi_1\rangle + \cdots + |\psi_k\rangle$. Then, the result of measuring observable O is that, the machine will collapse randomly to some outcome j with probability $|| \, |\psi_j\rangle \, ||$ and the new superposition of the machine will be $\frac{1}{|| \, |\psi_j\rangle \, ||}|\psi_j\rangle$.

2.2.2 Deciding the Time of Measurement

The decision as to when to measure an observable, is a crucial requirement, for the automaton to have the intended functionality. It is usually the case that, the machine is in a superposition of states, which means that, it essentially evaluates multiple paths simultaneously, and if the string on the tape has the desired structure, then the paths interfere either constructively or destructively. It is this aspect, that, in fact, distinguishes the qfa from classical automata, and is mainly responsible for their power. The point that is emphasized here, is that, the measurement is generally assumed to be done only after the interference mentioned above is guaranteed to have completed. Any measurement before that, would not guarantee proper functionality of the automaton. The instant of measurement, is also dependent on the string on the tape. In general, it is assumed that, there is some mechanism (that is not part of the unitary evolution of the automaton), which ensures that measurements are done only at appropriate moments. Strictly speaking, the definition of the automaton should include the specifications of the instants of measurement, along with the transition function, for it is the both that together define the functionality of the automaton under consideration.

2.3 The Possibility of Allowing Multiple Observables and Its Significance

The 2-qfa discussed above, allows measurements only with respect to one observable. This observable (that is referred to, here, as the primary observable) is a decomposition of the state space of the machine into subspaces constructed from non-halting states, halting states accepting the string on tape and halting states rejecting the string on tape. This measurement alone decides whether the machine is to further evolve or not, and in case of halting, it decides whether the string is accepted or not. Thus, it is this measurement that is used to define the languages accepted by a quantum automaton.

But, in general, it is possible to have different decompositions of the state space of the automaton (multiple observables). Such quantum automata possess sufficient power that, when distributed computation is attempted over these machines, additional power is not obtained, as shall be proved below. So, an interesting variant of the qfa is proposed below, wherein, multiple observables are allowed.

3 2qfa Allowing Measurements of Multiple Observables (2qfa-MOb)

3.1 Definition

A 2qfa-MOb is defined as a tuple

$$M = (Q, \Sigma, V, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$$

where Q,Σ,q_0,Q_{acc} and Q_{rej} are as before and the tape alphabet Γ being defined as before from Σ by including left and right end markers $\{\#, \$\}$. V is a set of unitary operators $V_\sigma : l_2(Q) \to l_2(Q)$ for every $\sigma \in \Gamma$ and $D : Q \to \{-1, 0, 1\}$. Moreover, it is understood that the transition function (δ) for the automata is constructed from V and D as in Eqn.1. Using V and D, instead of δ, is just a notational simplicity that provides convenience when dealing with distributed quantum automata. $\mathcal{O}$ is a set of observables for the automaton, excluding the primary observable (defined as the decomposition into subspaces, $Q_{acc} \oplus Q_{rej} \oplus Q_{non}$, where $Q_{non} = Q - (Q_{acc} \cup Q_{rej})$). Measurements can be performed with respect to any of the observables in $\mathcal{O}$. The instant of measurement for any observable is assumed to be well-defined as in the case of the usual 2-qfa.

3.2 Languages Accepted by 2qfa-MObs

For a given string $w \in \Sigma^*$, a tape x_w is constructed with length $|w|+2$, with $x_w(0) = \#$, $x_w(|w|+1) = \$$ and $x_w(i) = w_i$ for $1 \leq i \leq |w|$. The computation begins in the superposition $|q_0, 0\rangle$ and measurements can be made with respect to the primary observable and observables in $\mathcal{O}$, the instants of these measurements being defined, as mentioned above. It is the measurement of the primary observable that decides the acceptance of w. When any measurement of the primary observable results in a halting state, the computation halts. If the state is in the subspace constructed from Q_{acc}, w is accepted. Otherwise, the state is in the subspace constructed from Q_{rej}, and the string is rejected.

The computation can now be treated in the same manner as for a probabilistic machine. For instance, if input w results in *"accept"* with probability greater than 1/2, then w is an element of the language recognized by M, otherwise it is not. Just like probabilistic automata, restrictions such as running time and probability of error can be placed on the 2qfa-MOb as well.

4 Distributed 2qfa-MOb and Multiple-Choice 2qfa-MOb

4.1 Distributed 2qfa-MOb (D-2qfa-MOb)

4.1.1 Introduction

The D-2qfa-MOb is the quantum computational model corresponding to the classical distributed finite state automata. Different modes of acceptance can be defined here, in lines with the classical distributed automata.

4.1.2 Definition

A Distributed 2qfa-MOb (D-2qfa-MOb) is defined as a tuple,

$$M = (Q, \Sigma, V^1, V^2, \cdots, V^m, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$$

where Q,Σ,q_0,Q_{acc} and Q_{rej} are as before, the tape alphabet Γ being defined as before from Σ by including left and right end markers $\{\#, \$\}$ and $\mathcal{O}$ as in 2qfa-MOb is the set of observables excluding the primary observable, with the timings of measurements for every observable, assumed to be well-defined. $V^i = \{V^i_\sigma, \sigma \in \Gamma\}, i \in \{1, 2, \cdots, m\}$, where each $V^i_\sigma : l_2(Q) \to l_2(Q)$. Also, $D : Q \to \{-1, 0, 1\}$ and it is understood that transition functions (δs) for the automaton are constructed from V^i's and D as in Eqn.1, with V^i replacing V in the equation. Each transition function obtained is well defined for any i, since all V^i_σ are unitary. The component of the machine decides which transition function can be chosen at any moment.

During any moment of evolution, the machine is in exactly one of the components $i \in \{1, 2, \cdots, m\}$. If the symbol read by the head is σ, then the next stage of evolution operates V^i_σ on the current internal state of the machine and the machine goes to a new internal state. Different modes of acceptance can be defined for these distributed automata. The difference between the modes arises from the definition of allowed transitions from one component to another, for the machine. Acceptance is defined exactly in the same way as in 2qfa-MOb with respect to the outcome of measurements of the primary observable.

4.1.3 Modes of Acceptance

There are four possible modes of acceptance:

- $*$ - mode - The transition from component i to some other component j can occur at any arbitrary stage of the evolution.

- $= k$ - mode - The transition from component i to some other component j occurs after exactly k transitions using the V^is.

- $\geq k$ - mode - The transition from component i to some other component j occurs after at least k transitions using the V^is.

- $\leq k$ - mode - The transition from component i to some other component j occurs before $k + 1$ transitions using the V^is.

The language accepted by a D-2qfa-MOb M using mode of acceptance $\alpha, \alpha \in \{*, = k, \geq k, \leq k | k \geq 1\}$ is denoted by $L(M, \alpha)$. Note that, in some cases the automata constructed is such that, all strings belonging to L are accepted definitely, whereas strings not in L are accepted with a finite probability bounded by a small value (Automata with one-sided bounded error). In such cases, the bound is also included in the notation as $L(M, \alpha, \text{bound})$.

4.2 Multiple-Choice 2qfa-MOb (MC-2qfa-MOb)

4.2.1 Introduction

The Multiple-Choice-2qfa-MOb (MC-2qfa-MOb) is the quantum computational model corresponding to the classical non-deterministic finite state automata(NFA). ie., There are choices in the evolution at any stage, and one of the choices can be chosen non-deterministically.

4.2.2 Definition

A MC-2qfa-MOb is defined as a tuple,

$$M = (Q, \Sigma, V, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$$

where Q,Σ,q_0,Q_{acc} and Q_{rej} are as before, the tape alphabet Γ being defined as before from Σ by including left and right end markers $\{\#, \$\}$ and $\mathcal{O}$ as in 2qfa-MOb is the set of observables excluding the primary observable, with the timings of measurements for every observable, assumed to be well-defined. Now, $V = \{V_\sigma | \sigma \in \Gamma\}$. Each V_σ is now, a finite multiset of unitary operators, unlike the 2qfa-MOb wherein, V_σ is just a single operator. ie., $V_\sigma = \{V_{(\sigma,i)}, i \in \mathbb{N}\}$, and $V_{(\sigma,i)} : l_2(Q) \to l_2(Q)$.

As usual, $D : Q \to \{-1, 0, 1\}$ and it is understood that transition functions (δs) for the automata are constructed from V and D as in Eqn.1, with any $V_{(\sigma,i)}$ replacing V_σ in the equation. Any of the transition functions obtained can be used for the evolution of the automaton.

At any moment of evolution, if the symbol at the head is σ, then one of the operators in V_σ is chosen non-deterministically and is operated on the current internal state.

4.2.3 Choice of an Operator at an Instant

It should be noted that V_σ is a multiset since repetition of operators is allowed. The number of times a particular operator recurs in the set affects its probability of choice from the whole set. Suppose, a particular operator $\mathcal{V}$ occurs n times in V_σ, which is to say, $card(\{i | V_{(\sigma,i)} = \mathcal{V}\}) = n$. Then, the probability that $\mathcal{V}$ is chosen for evolution in the next stage is equal to $\frac{n}{card(V_\sigma)}$. (Here, $card(X)$ denotes the cardinality of the set or multiset X). Note that, for any MC-2qfa-MOb $M(Q, \Sigma, V, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$, an equivalent MC-2qfa-MOb $M'(Q, \Sigma, V', D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$ with $card(V'_\sigma) = card(V'_\beta)$ for all $\sigma, \beta \in \Gamma$ (ie., all V'_σ having the same number of operators) can be constructed trivially, in the following way:

Let N be the L.C.M. (Least common multiple) of all the cardinalities, $card(V_\sigma)$. For each $\sigma \in \Gamma$, find $replication(\sigma) = \frac{N}{card(V_\sigma)}$ and construct V'_σ from V_σ by replicating each operator $\mathcal{V} \in V_\sigma$, $replication(\sigma)$ times. (If the operator occurs n times, it is replicated $n \times replication(\sigma)$ times). Now, $card(V'_\sigma) = N$, for any $\sigma \in \Gamma$. Also, this operation leaves the probability of selection of any operator in V_σ unchanged. This is because, for any $\mathcal{V}$ occurring n times in V_σ, the probability of selection of $\mathcal{V}$ from V'_σ is given by $\frac{n \times replication(\sigma)}{N} = \frac{n}{card(V_\sigma)}$. Hence, the probability remains the same. So, M' is equivalent to M and also has the same number of operators in each

$V'(\sigma)$.

From now on, it is assumed without loss of generality that, any MC-2qfa-MOb M has the same number of operators in each of its V_σs, and this number is referred to as the choice dimensionality (number of possible choices) of M. Also, it is understood that when simply written as 2qfa-MOb, it means that, there is no choice for the selection of operators. The language accepted by a MC-2qfa-MOb M is denoted by $L(M)$. In case of automata with one-sided bounded error, the bound is also included in the notation as $L(M, \text{bound})$.

5 Equivalence of MC-2qfa-MOb and 2qfa-MOb

5.1 Nature of Equivalence

In this section, the MC-2qfa-MOb is shown to be equivalent to the 2qfa-MOb. An exact equivalence is got in the case when the choice dimensionality (number of operators to choose from) is of the form 2^n, for some $n \in \mathbb{N}$. When it is not of such a form, then an equivalence can be obtained arbitrarily accurately (with bounded error).

5.2 Derivation of the Result

Theorem 5.2.1 *Any* MC-2qfa-MOb $M(Q, \Sigma, V, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$ *can be reduced to an equivalent* 2qfa-MOb $M'(Q', \Sigma, V', D', q_0', Q_{acc}', Q_{rej}', \mathcal{O}')$ *with single choice for evolution operators.*

Proof: M' is constructed as follows:

Let N be the choice dimensionality of M. Two cases are considered.

5.2.1 Case-I: N is of the form 2^n, for some $n \in \mathbb{N}$ - Exact Equivalence

Construction:

Consider the set $C = \{1, 2, \cdots, N\}$. The Hilbert space constructed with elements in C ($l_2(C)$), is denoted by Ω. Now $Q' = Q \otimes C$. ie., Any valid state of M' is of the form $\sum_{q,i} a_{(q,i)}|q, i\rangle, q \in Q, i \in C, \sum a^2_{(q,i)} = 1$. (Here, the $a_{(q,i)}$'s denote the amplitudes, and $|q, i\rangle$ stands for $|q\rangle \otimes |i\rangle$).

$D'(|q, i\rangle) = D(q)$ for all $q \in Q, i \in C$.

$Q'_{acc} = Q_{acc} \otimes C$.

$Q'_{rej} = Q_{rej} \otimes C$.

$q_0' = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} (|q_0, i\rangle)$.

$\mathcal{O}' = (\mathcal{O} \otimes C) \cup \omega$, where ω is the observable $(Q \otimes |0\rangle) \oplus (Q \otimes |1\rangle) \oplus \cdots \oplus (Q \otimes |N\rangle)$ which is to say that, all the original observables are preserved here by expanding the subspaces in the decompositions to the whole configuration space corresponding to Q', with the addition of a new observable, ω which measures the $|i\rangle$ portion of the superposition $\sum_{q,i} a_{(q,i)}|q, i\rangle$. The original timing information of the observables in $\mathcal{O}$ is still carried over to the corresponding observables in $\mathcal{O}'$. As far as measuring

ω is concerned, a measurement is made after every step in the evolution.

Now, let

$$E(1) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Construct $E(i)$s recursively from $E(i-1)$s using,

$$E(i) = \frac{1}{\sqrt{2}} \begin{pmatrix} (E(i-1)) & (E(i-1)) \\ (E(i-1)) & (-E(i-1)) \end{pmatrix}$$

The $E(i)$ are matrices of order 2^i. It is easily seen that, for any i, $E(i)$ is unitary (since $E(i) \times E(i)^T = I_{2^i}$, where I_{2^i} is the identity matrix of order 2^i). Also, $E(i)$ has the property that, each element in it is either $\frac{1}{\sqrt{2^i}}$ or $-\frac{1}{\sqrt{2^i}}$.

Now, the V' is constructed from V and $E(n)$ (Note that $N = 2^n$) as follows: If the internal state of the automaton is $|q, i\rangle$ and the symbol on the head is σ, then the evolution on applying V'_σ should be such that,

$$V'_\sigma(|q, i\rangle) = V_{(\sigma,i)}(|q\rangle) \otimes E(n)(|i\rangle)$$

The construction is as below when the states and operators are expressed explicitly in matrix form. Let $Q = \{q_1, q_2, \cdots, q_s\}$, $s = card(Q)$. The amplitudes of the the current superimposition of states of the machine can be expressed as a column vector of the amplitudes $a_{(q,i)}$s as

$$(a_{(q_1,1)}, a_{(q_2,1)}, \cdots, a_{(q_s,1)}, a_{(q_1,2)}, a_{(q_2,2)}, \cdots, a_{(q_s,N)})^T.$$

Now, if

$$E(n) = \begin{pmatrix} a_{11} & a_{12} & . & . & . & a_{1N} \\ a_{21} & a_{22} & . & . & . & a_{2N} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{N1} & a_{N2} & . & . & . & a_{NN} \end{pmatrix}$$

then

$$V'_\sigma = \begin{pmatrix} (V_{(\sigma,1)}) \times a_{11} & (V_{(\sigma,2)}) \times a_{12} & . & . & . & (V_{(\sigma,N)}) \times a_{1N} \\ (V_{(\sigma,1)}) \times a_{21} & (V_{(\sigma,2)}) \times a_{22} & . & . & . & (V_{(\sigma,N)}) \times a_{2N} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ (V_{(\sigma,1)}) \times a_{N1} & (V_{(\sigma,2)}) \times a_{N2} & . & . & . & (V_{(\sigma,N)}) \times a_{NN} \end{pmatrix}$$

The construction of $V'(\sigma)$ is very similar to that of a tensor product and it can be easily verified that V'_σ is unitary using the fact that, all V_σs and $E(n)$ are unitary.

Proof of equivalence:

The proof rests on the fact that, the $|i\rangle$ component of the internal state $|q, i\rangle$ for M' simulates the operation of $V_{(\sigma,i)}$ on the current state $|q\rangle$ on M, when the symbol

read by the head is σ. Consider any intermediate step in the evolution of M. Let the symbol read on one of the paths at instant $t-1$ be σ and the state be q. Now, one of the matrices in V_σ is chosen non-deterministically. Without loss of generality, we assume that, $V_{(\sigma,j)}$ was chosen. Now, the new state of the machine becomes $V_{(\sigma,j)}(|q\rangle) = |v\rangle$ (say), the head moves in the direction $D(|v\rangle)$ and all observables that can be measured at that instant, as per the timing specification in O can be measured. If the primary observable is measured, then the halting/non-halting and acceptance/rejection decisions are made. The time now becomes t. If not halted, the evolution proceeds in the same manner for time instant t.

The same situation translates into the construction and functionality of M' as follows. At time $t-1$, the symbol read on the path is σ and the state of the automaton is

$$r(t-1) = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} (|q,i\rangle).$$

The form above corresponds to the state $|q\rangle$ of M, as will be evident, once the evolution is understood. Now, a measurement of ω is done. This collapses state $r(t-1)$ to the state $|q,j\rangle$ with the same probability $\frac{1}{N}$ as in the case of M. Now, V'_σ is applied on the state $|q,j\rangle$ to give

$$V'_\sigma(|q,j\rangle) = V_{(\sigma,j)}(|q\rangle) \otimes E(n)(|j\rangle) = |v\rangle \otimes \sum_{i=1}^{N} (|i\rangle)$$

which takes the machine to the state

$$r(t) = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} (|v,i\rangle)$$

which corresponds to the state $|v\rangle$ of M in the same way as before. This consistent correspondence of the states of M' to those of M is because, $E(n)$ is such that, for any $k \in C$ and correspondingly $|k\rangle \in \Omega$, $E(n)(|k\rangle) = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} (|i\rangle)$. The head moves in the direction $D'(|v,i\rangle) = D(|v\rangle)$ for any i. The measurements mentioned above are made with respect to corresponding observables in M' and as seen, the primary observable of M' is defined consistently in terms of Q'_{acc} and Q'_{rej}. Also, the initial state q'_0 is such that a measurement of ω initially would set the state to $|q_0,j\rangle$ for some j. Thus, M' exactly simulates the behaviour of M, and hence the languages accepted by them are exactly equivalent (the exactness here, also, includes acceptance with one-sided bounded error, since the probabilities of the choice of operators have been taken care of, accurately).

5.2.2 Case-II: N is NOT of the form 2^n, for any $n \in \mathbb{N}$ - Equivalence with bounded error

Construction:

The equivalence in the previous case was proved when N is of the form 2^n. The matrices $E(i)$ defined above have order 2^i. Matrices with similar properties (unitarity and equal distribution of amplitudes), but orders not of the form 2^i cannot be constructed. Hence, when the choice dimensionality is not of the form 2^n, equivalence cannot be proved in exactly the same way before. Here, equivalence

is proved allowing small violations in the probability of choice of operators. Such violations in anyway do not affect the class of languages, accepted by automata with certainty. Only the probability of acceptance or rejection, in the case of automata with bounded error, is affected. This deviation can be made arbitrarily small, as shall be shown below. In this way, M' can simulate M as accurately as possible.

The construction is done by choosing a large number $N' = 2^m$, for some $m \in \mathbb{N}$, such that $N' > N$. Let f be the quotient and g the remainder when N' is divided by N. For each $\sigma \in \Gamma$, replicate $V_{(\sigma,i)}$, for every i, f times. Then, reinclude the first g operators, $\{V_{(\sigma,i)} | 1 \leq i \leq g\}$. Let the set of operators now be denoted $V"_\sigma$. Consider the automata $M" = (Q, \Sigma, V", D, q_o, Q_{acc}, Q_{rej}, \mathcal{O})$. The number of operators in each of $V"_\sigma$ is now 2^m. Then, the construction is done exactly as in Case-I and the automata M' is got. Now, it is clear that, M' accepts the same class of languages accepted by M, and only in the case of languages accepted with finite one-side bounded error, the probabilities of acceptance or rejection might slightly differ.

The probability of choosing the ith operator, $V_{(\sigma,i)}$ from V_σ at any moment is $\frac{1}{N}$, whereas the probability of choosing the same operator from $V"$ is equal to either $\frac{f}{N'}$ or $\frac{f+1}{N'}$. As shown below, the difference in the two probabilities can be made arbitrarily small, by appropriately choosing N', and hence, $M"$ (and hence M') can be made to simulate M arbitrarily accurately.

Choice of N':

Suppose the bound that we impose on the difference in the probabilities is ε. ie.,

$$\frac{f+1}{N'} - \frac{1}{N} < \varepsilon.$$

To achieve this bound, N' is chosen as follows. Find numbers m and λ, with $\lambda \leq \varepsilon$ such that,

$$\frac{1}{2^m} > \lambda > (\frac{N-1}{N})\frac{1}{2^m}$$

m and λ can be found for any given ϵ, since it is always possible to choose as high an m as possible (corresponding to lower and lower λ values). Now, $N' = 2^m$. This choice of m and λ gives

$$\frac{1}{\lambda} > 2^m (= N') > (\frac{N-1}{N})\frac{1}{\lambda} \Rightarrow N - NN'\lambda < 1.$$

Since the remainder g is non-zero (at least one), we have

$$1 \leq N' - Nf < N$$

$$\Rightarrow N' - Nf > N - NN'\lambda$$

$$\Rightarrow N(f+1) < N'(1 + N\lambda)$$

which reduces to,

$$\frac{f+1}{N'} - \frac{1}{N} < \lambda \leq \varepsilon.$$

In an exactly similar way, the choice of N' for the bound,

$$\frac{1}{N} - \frac{f}{N'} < \varepsilon$$

can be done. In order for both the bounds to hold good, we choose the bound corresponding to the lesser of the two differences, and choose N' based on that.

The bound on the probability difference translates directly to the bound on the difference in the probabilities of one-sided rejection. Hence, if the choice dimensionality is not of the form 2^n, an equivalence with bounded error can be obtained.

5.3 Result

Thus, the MC-2qfa-MOb are reducible to the 2qfa-MOb, exactly when the choice dimensionality is of the form 2^n and arbitrarily closely otherwise. ie., for every MC-2qfa-MOb M, there exists a 2qfa-MOb M' such that, $L(M) = L(M')$. If M has acceptance with bounded error (bound ε), then a 2qfa-MOb M" can be constructed such that, $L(M, \varepsilon) = L(M", \vartheta)$, such that, $|\varepsilon - \vartheta| < \varsigma$, for any arbitrarily small ς.

6 Equivalence of D-2qfa-MOb and MC-2qfa-MOb

6.1 Theorem:

For any D-2qfa-MOb M, there exists a MC-2qfa-MOb M', such that, $L(M, \alpha) = L(M')$, for any $\alpha \in \{*, = k, \leq k, \geq k\}$. In the case of M being a D-2qfa-MOb with one-sided bounded error ε for acceptance, then, a MC-2qfa-MOb M' can be constructed such that, $L(M, \alpha, \varepsilon) = L(M', \varepsilon)$

6.1.1 Proof:

The result is proved by reducing a D-2qfa-MOb M to a MC-2qfa-MOb M', for each mode of transition, as below:
Case-I: $\alpha = *$

Let $M = (Q, \Sigma, V^1, V^2, \cdots, V^m, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$. The equivalent MC-2qfa-MOb is constructed as follows. $M' = (Q', \Sigma, V', D', q'_0, Q'_{acc}, Q'_{rej}, \mathcal{O}')$.
Let $C = \{1, 2, \cdots, m\}$. Let $\mathcal{H} = l_2(C)$ denote the Hilbert space constructed using the elements of C as basis. Also, let $s = card(Q)$.

$Q' = Q \otimes \mathcal{H}$. ie., Q' can be represented as $\{[q, i] | 1 \leq i \leq m, q \in Q\}$. The amplitudes of the current superimposition of states of the machine can be expressed as a column vector of amplitudes,
$(a^1_{q_1}, a^1_{q_2}, \cdots, a^1_{q_s}, a^2_{q_1}, \cdots, a^m_{q_s})^T$, where $Q = \{q_i | 1 \leq i \leq s\}$ and $a^i_{q_j}$ is the amplitude corresponding to the internal state $q_j \in Q$, when the machine is in component i. Since the machine can be in only one of the components at any instant, the amplitudes corresponding to all other components are zeroes.

$D'([q, i]) = D(q)$, for all $q \in Q$ and $i \in C$.

$q'_0 = [q_0, 1]$.

$Q'_{acc} = \{[q, i] | q \in Q_{acc}, i \in C\}$.

$Q'_{rej} = \{[q, i] | q \in Q_{rej}, i \in C\}$.

$\mathcal{O}'$ is obtained as follows. For an observable (decomposition) $O \in \mathcal{O}$ with $O = S_1 \oplus S_2 \oplus \cdots \oplus S_l$, for some l, then an observable $O' = (S_1 \otimes \mathcal{H}) \oplus (S_2 \otimes \mathcal{H}) \oplus \cdots \oplus (S_l \otimes \mathcal{H})$ is included in O', with the same timing specifications as that of O.

To construct V', the various possible transitions among the components of V are represented as permutations over the set C . A permutation over C is a one-one mapping $P : C \to C$. It can be conveniently represented as a m-tuple $(P(1), P(2), \cdots, P(m))$. For any given mode of transition in M, there are many allowed permutations. Let P_{al} denote the set of all allowed permutations over C for the mode of transition defined in M. As shown below, each allowed permutation corresponds to a choice in the operators in V'. When the machine is in component i, the head reads symbol σ and the permutation (m-tuple) $\mathcal{P} = (P(1), P(2), \cdots, P(m))$ is chosen, the following is done.

- The operator V_σ^i is used to transform the current internal state of the machine. This is achieved through the function matrices described below.
- The machine then transits to the component $P(i)$ in the next step. This is achieved through the transition matrices described below.

Let $\mathcal{P} = (P(1), P(2), \cdots, P(m))$ be an allowed permutation over C, for the mode of transition in M. $\mathcal{P}$ induces a transition matrix $T(\mathcal{P})$ which specifies the transformation of the state in M' corresponding to the change of component in M. Also, for each $\sigma \in \Gamma$, there is a function matrix $F(\sigma)$ which specifies the evolution of the state in M', when the symbol read is σ. $F(\sigma)$ is essentially derived from the Vs of the various components in M..

To understand the construction of these matrices, it is helpful to think of them as $m \times m$ grids of cells, with matrices of order $s \times s$ in each cell. Let $I(s)$ and $O(s)$ denote identity and zero matrices of order $s \times s$ respectively. The grid cell at the kth row and l^{th} column is denoted by $C(k, l)$.

Now, $F(\sigma)$ is defined as the matrix of cells $C_F(k, l), 1 \leq k, l \leq$ given below.

$$C_F(k,l) = \begin{cases} O(s) & k \neq l \\ V_\sigma^k & k = l \end{cases}$$

$T(\mathcal{P})$ is defined as the matrix of cells $C_T(k, l), 1 \leq k, l \leq$ given below.

$$C_T(k,l) = \begin{cases} I(s) & P(l) = k \\ O(s) & \text{otherwise} \end{cases}$$

For instance, the construction of the matrices are shown for the case $m = 4$, $s = 2$ and $\mathcal{P} = (4, 1, 3, 2)$ (Take component 1 to component 4, component 2 to component 1, etc...) is shown below.

$$F(\sigma) = \begin{pmatrix} (V_\sigma^1) & (O(2)) & (O(2)) & (O(2)) \\ (O(2)) & (V_\sigma^2) & (O(2)) & (O(2)) \\ (O(2)) & (O(2)) & (V_\sigma^3) & (O(2)) \\ (O(2)) & (O(2)) & (O(2)) & (V_\sigma^4) \end{pmatrix}$$

$$T(\mathcal{P}) = \begin{pmatrix} (O(2)) & (I(2)) & (O(2)) & (O(2)) \\ (O(2)) & (O(2)) & (O(2)) & (I(2)) \\ (O(2)) & (O(2)) & (I(2)) & (O(2)) \\ (I(2)) & (O(2)) & (O(2)) & (O(2)) \end{pmatrix}$$

ie.,

$$T(\mathcal{P}) = \begin{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{pmatrix}$$

$$T(\mathcal{P}) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

since

$$I(2) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad and \quad O(2) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Now suppose the machine M was in component 2 in state $(a_1, a_2)^T$, where a_1, a_2 are the probability amplitudes corresponding to states q_1, q_2, the only states in Q and the current symbol read is σ. Then the state of the machine M' is the column vector $\alpha = (0, 0, a_1, a_2, 0, 0, 0, 0)^T$. Now, the choice of the permutation $\mathcal{P} = (4, 1, 3, 2)$ would mean that, the component should change to 1 after the application of V_σ^2. This is essentially accomplished by the operation $T(\mathcal{P}) \times (F(\sigma) \times \alpha)$. Suppose $V_\sigma^2 \times (a_1, a_2)^T = (b_1, b_2)$, then the above operation $(T \times F \times \alpha)$ would result in the state $\beta = (b_1, b_2, 0, 0, 0, 0, 0, 0)^T$ in consistence with the description of the correspondence of states in M' to states in M.

Since V_σ^i is unitary for all i, $F(\sigma)$ is also unitary $(F(\sigma)F^T(\sigma) = I)$. Also, it is evident that, the transition matrices are unitary too (since there is exactly one 1 in each row (column) with the rest being zeroes, the rows (columns) are linearly independent). Now,

$$V'_\sigma = \{T(\mathcal{P}) \times F(\sigma) \mid \mathcal{P} \in P_{al}\}$$

Thus, M' simulates the function of M, provided the allowed permutations in M' clearly reflect the mode of transition of M. In the "*-mode" of transition, since a transition from the current component i to some component j can occur at any instant, P_{al} includes all permutations over C. ie., all permutations are valid, since any permutation specifies a valid transition between components for M. Hence, when V' is constructed as above, using the permutations in P_{al}, M' simulates M exactly. Since the event of choosing any of the matrices from V'_σ is equally likely, all

permutations are equally likely, which means that, the probability of switching from the current component i to a component j is the same as that of switching to any other component k (There are equal number of permutations that take component i to component j, as there are for transitions from i to k). Also, it is easy to see that, acceptance in M' is defined in exact correspondence to observables in M, and the initial state is defined consistently.

Thus, M' is exactly equivalent in behaviour to M, and hence
$L(M, "*-mode") = L(M')$.
Case-II: $\alpha = " = k"$

The proof for the "=k mode" is very similar to that of the "*-mode". The derivation is looked in the following way. The system M, is looked upon as an equivalent system
$M" = (Q, \Sigma, V^{11}, V^{12}, \cdots, V^{1k}, V^{21}, \cdots, V^{mk}, D, q_0, Q_{acc}, Q_{rej}, \mathcal{O})$, where for all i, V^{ik} of $M"$ is a component created by an exact replication of V^i of M, and now, the transitions are to be forced between components 11 and 12, etc... whereas, there is a choice in transition between ik and $j1$, for any $i, j \in C$. This is accomplished by constructing the MC-2qfa-MOb M' in the same way as before, except for a change in P_{al}. All permutations are not allowed, since as said above, some transitions are to be forced. Hence, only permutations that translate ij to $i(j+1)$, if $j < k$ and to $m1$ otherwise, are allowed. The transition matrices corresponding to these permutations are constructed and V'_σ are constructed as before, by premultiplying the function matrix with these matrices. Then, M' simulates M exactly, and hence, $L(M, " = k - mode") = L(M')$.
Cases-III and IV: $\alpha = " \leq k"|" \geq k"$

The proof for these cases is exactly similar to that of the previous cases. The key to the proof lies in the fact that, the mode of transition in M can be exactly simulated in M' by appropriately defining P_{al} by identifying the permutations that specify valid transitions of components for M.

6.2 Result

Hence, any D-2qfa-MOb is exactly reducible to a MC-2qfa-MOb, for any of its modes of acceptance. Combining this result with the equivalence of MC-2qfa-MOb and 2qfa-MOb, the following result is also true. For any D-2qfa-MOb M, there exists a 2qfa-MOb M', such that, $L(M, \alpha) = L(M')$, for any $\alpha \in \{*, = k, \leq k, \geq k\}$. In the case of M being a D-2qfa-MOb with one-sided bounded error ε for acceptance, then, a 2qfa-MOb M' can be constructed with, $L(M, \alpha, \varepsilon) = L(M', \vartheta)$ such that, $|\varepsilon - \vartheta| < \varsigma$, for any arbitrarily small ς.

7 D-1qfa-MOb

In a way exactly similar to 2-qfa, distributed and multiple choice 1-qfa can also be defined. The corresponding results also hold good here, once multiple observables are allowed. Thus, both D-1qfa-MOb and MC-1qfa-MOb are equivalent in power to the 1qfa-MOb with single choice, and acceptance with certainty, whereas, in the case

of automata with bounded error, D-1qfa-MOb and MC-1qfa-MOb can simulate the behaviour of 1qfa-MOb arbitrarily closely.

8 Conclusion

Thus, in this work, first, quantum automata were generalized to include multiple observables. Then, distribution was attempted. Multiple Choice quantum automata and Distributed quantum automata were considered, in lines with classical NFA and distributed finite state automata, and it is found that, the power of these automata are the same as that of the simple quantum automata with multiple observables and single choice. But the question as to whether allowing multiple observables results in increase in power (whether the 2qfa-MOb are more powerful than 2qfa) has not been answered in this work.

References

[1] A. Kondacs and J. Watrous. On the power of quantum finite state automata. *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 66-75, 1997.

[2] P. Shor. Algorithm for quantum computation: discrete logarithm and factoring. *35th Annual Symposium on Foundations of Computer Science*, pages 124-134, 1994.

[3] L. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 212-219, 1996.

[4] K. Krithivasan, M. Sakthi Balan and Prahlad Harsha. Distributed processing in automata. *International Journal of Foundations of Computer Science*, Vol 10, No.4, pages 443-464, 1999.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 30 - 49, 2004.

A Membrane Systems Explanation of Semantic Change*

Gemma Bel Enguix

Research Group on Mathematical Linguistics, Universitat Rovira i Virgili
Pl. Imperial Tàrraco, 1, 43005 Tarragona, Spain,
and
Department of Computer Science, University of Milan-Bicocca
Via Bicocca degli Arcimboldi, 8, 20126 Milan, Italy
gbe@astor.urv.es

M. Dolores Jiménez López

Research Group on Mathematical Linguistics, Universitat Rovira i Virgili
Pl. Imperial Tàrraco, 1, 43005 Tarragona, Spain
mdjl@astor.urv.es

Abstract

An application of membrane systems to natural languages study is presented. Specifically, we show how *semantic change* can be explained by using a new variant of membrane systems: Dynamic Meaning Membrane Systems (DMMS). By using DMMS, we explain the three basic types of changes in meaning -this is, *broadening*, *narrowing* and *shift*. Finally, we relate the membrane systems' application to language evolution with the suggested application of the so-called *cultural grammar systems* to the same topic. Collaboration between the two frameworks may provide a useful formalism that, due to its naturalness and simplicity, might offer interesting results in a discipline traditionally far away from any formalization.

1 Introduction

An application of membrane systems to natural languages study is presented. Specifically, in this paper, we show how *semantic change* can be explained by using a new variant of membrane systems. An initial application of membrane systems to linguistics was introduced in [2]. The most important intuition for translating this natural computing model to natural languages is that membranes can be understood as *contexts*. Contexts may be different words, persons, social groups, historical periods, languages. They can accept, reject, or produce changes in elements they have inside.

*This research has been supported by a Marie Curie Fellowship of the European Community programme *Human Potential (IHP)* under contract number HPMF-CT-2002-01582 and by a Marie Curie European Reintegration Grant (ERG).

At the same time, contexts/membranes and their rules evolve, that is, change, appear, vanish, etc. All these features make of membrane systems an attractive model to deal with natural language.

If there is something obvious is that languages change. Languages are continuously in flux. Changes in languages will occur in all realms of linguistic organization: the pronunciation, the sound system, the morphology, the lexicon, the semantics, the syntax. A language can change so much as to become a different language or evolve into several different languages. Related languages, which belong to the same group or family, were once the same language. This is, they are derived, due to the operation of 'linguistic change' over long periods of time, from a single, earlier ancestor language. Taking into account that every dimension of language evolves, it is necessary to study language change at different levels. In this paper we will concentrate only on *semantic change*. It has been said that semantics is most easily and radically affected by change than phonology, morphology and syntax. Paradoxically, even though being the dimension of language that changes most easily, semantic evolution is the area of historical linguistics that is least well understood. However, in the literature of diachronic linguistics, taxonomies of semantic change can be found as well as generalities about the mechanics of meaning evolution. For a general introduction to historical linguistics, see [11, 6, 15, 14, 8, 17, 1].

In this paper we will try to approach semantic change in a completely new way. We will provide a *membrane system model* to explain semantic change and we will suggest the possibility of combining this framework with an already defined variant of eco-grammar systems, the so-called *cultural grammar systems* ([9], [10]). Collaboration between the two frameworks may provide a useful formalism that, due to its naturalness and simplicity, might offer interesting results in a discipline traditionally far away from any formalization.

In what follows, we first introduce Linguistic Membrane Systems (LMS) as a general framework that can be used to model in terms of membrane systems different linguistic issues. By modifying some definitions of LMS we obtain Dynamic Meaning Membrane Systems (DMMS), a dynamic model for semantics that can be used in the explanation of semantic change in natural language. By using DMMS, we explain the three basic types of changes in meaning -this is, *broadening, narrowing and shift*. We conclude the paper by briefly relating the membrane systems' application to language evolution with the suggested application of the so-called *cultural grammar systems* to the same topic (cf. [9, 10]).

2 Linguistic Membrane Systems

As generative devices for formal languages, membrane systems do not take into account the problem of meaning in their output. In the initial formalization, membranes are aseptic contexts, this is, contexts which do not have anything to do with the derivation carried out in the system. However, in linguistics contexts matter. Therefore, if membrane systems have to be applied to linguistics, the first required adaptation is to make them able to deal with meaning. Thus, we need to introduce several new notions in order to define Linguistic Membrane Systems (LMS). One of

the first notions we need is the notion of the *domain* of a membrane. The domain is to be understood as the set of words, statements, ideas and elements a membrane is able to work with in a given state of the computation. The domain is an active context. During computation a membrane receives some elements which are accepted by its domain and some which are not. We have to explain what to do with elements that are not part of the configuration of the membrane. Another interesting feature is the fact of working with several alphabets, this can be used as a tool for explaining adaptation of a linguistic element from a context to another one. In what follows we will define one by one all those useful concepts for the application of membrane systems to linguistics:

- **Alphabets**: A LMS has one or more *alphabets*, which can change or evolve during the computation. Each alphabet evolves independently.
- **Domains**: The *domain* D of a membrane is the definition of the symbols it accepts. Domains are related to one or more alphabets (for example, the domain DM_n can be the union of two alphabets, $V_m \cup V_j$). The domain of the skin membrane[1] is the union of the domains of its internal membranes. Several membranes of the same system can have the same domain.
- **Adaptation of elements which are not in the domain**:
  - *Function of transposition.* The function h, called function of *transposition*, establishes a correspondence between symbols placed in different membranes. For instance, the function $h(M_n \leftrightarrow M_m)$ establishes the correspondence between symbols belonging to the membrane M_n and symbols belonging to the membrane M_m. The rules of this function have the following form: $h(M_n \leftrightarrow M_m) = \{a_{in_{M_n}} \leftrightarrow \alpha_{in_{M_m}}, \ldots, b_{in_{M_n}} \leftrightarrow \beta_{in_{M_m}}\}$. They are called *rules of transposition.* Sometimes, transposition rules have the symbol "$\rightarrow$" instead of "$\leftrightarrow$". They are *non-returning transposition rules.*
  - *Non-adapted symbols.* The subscript $_i$ attached to an element in a membrane means that this element is not accepted by the domain of the membrane which it belongs to. If a is not included in the domain of M_m and the rule $h(M_n \leftrightarrow M_m) = a_{in_{M_n}} \leftrightarrow \alpha_{in_{M_m}}$ does not exist, then $a_{in_{M_m}} = a_i$. Elements marked with $_i$ are not taken into account as output of the membrane system when computation stops.

One of the most interesting points of the theoretical formalization of membrane systems is the concept of *evolution*: the system evolves. Notice that this is a very important feature in order to fulfill our goal because natural languages –as membrane systems– are constantly evolving. However, we need to construct systems able to change in any configuration and this is the case of LMS where not only membranes evolve, but alphabets, domains and rules evolve as well, as shown in what follows:

[1] In a membrane structure –understood as a hierarchical arrangement of membranes– the *skin membrane* is the one that separates the system from its environment.

- Evolution of *alphabets:*

  1. Addition of some symbols: for example, for $V_1 = \{1, 2, 3\}$, the rule $ADD\ \{4\}\ TO\ V_1$ increases the alphabet so as $V_1 = \{1, 2, 3, 4\}$.
  2. Deletion of some symbols: for example, for $V_1 = \{1, 2, 3\}$, the rule $DEL\ \{3\}\ FROM\ V_1$ decreases the alphabet so as $V_1 = \{1, 2\}$.

- Processes for the evolution of *domains*, which are called *variable domains*:

  1. Addition of new symbols to the alphabets belonging to the domain. For example, for $V_1 = \{1, 2, 3\}$ and $D_{M_n} = V_1$, the rule $ADD\ \{4\}\ TO\ V_1$ increases the domain.
  2. Deletion of some symbols from the alphabets belonging to the domain. For example, for $V_1 = \{1, 2, 3\}$ and $D_{M_n} = V_1$, the rule $DEL\ \{3\}\ FROM\ V_1$ decreases the domain.
  3. Addition of new alphabets to the domain. For example, for $V_1 = \{1, 2, 3\}$, $V_2 = \{a, b, c\}$ and $D_{M_n} = V_1$, the rule $ADD\ V_2\ TO\ D_{M_n}$ has as a result $D_{M_n} = V_1 \cup V_2$.
  4. Deletion of some alphabets from the domain. $DEL\ V_1\ FROM\ D_{M_n}$, applied to $D_{M_n} = V_1 \cup V_2$, has as a result $D_{M_n} = V_2$.

3 Dynamic Meaning Membrane Systems

3.1 Some Semantic Definitions

Croft points out in [5] that it is possible to give two different definitions of meaning, based on "*the distinction between a language as a population of utterances produced by a speech community, and a grammar as an individual speakers knowledge about the conventions of the speech community*". Such definitions are the following:

- The **community's meaning** of a linguistic form -a lingueme- is the lineage of replication of its use, in their full encyclopedic, contextual value.

- The **individual's meaning** of a lingueme is a mental structure that emerges from the individual's exposure to (necessarily partial) lineages of the community's meaning, including of course the use of the lingueme by that same individual.

In order to adapt semantic definitions to Dynamic Meaning Membrane Systems (DMMS) we take the notion of lingueme referred to in the above definitions. *Lingueme* is a linguistic unit, an utterance. No matter if it is a word, a sentence or a discourse. A lingueme, in the first state, has no meaning. From the concept of lingueme we infer another concept, *semanteme*, which is a semantic unit. The process of meaning assignment is given by the application of a semanteme to a lingueme. Such application is called *convention.*

Since linguemes and semantemes will be two notions that we will use in the definition of DMMS, we have to define the following two alphabets:

- $V = \{a, .., z\}$, the set of linguemes.
- $\Sigma = \{\alpha, .., \omega\}$, the set of semantemes.

And finally, we formally define the concepts of *individual meaning* and *community's meaning* as follows:

- The *individual's meaning*, I, of an utterance is an application of the function $F(\Sigma \to V)$, this is the application of a semanteme to a lingueme. The individual's meaning of a lingueme a in membrane M_m, $I(a)M_m$ is given by the function $F(\Sigma \overset{a}{\to} V)M_m$.
- The *community's meaning*, K, of an utterance is the composition of individual meanings in each step of the computation of a system. In each step, community's meaning of an utterance a is $K(a) = \bigcup_{i+1}^{m} I(a)M_i$.

3.2 Defining DMMS

Definitions of membrane systems can be found in [12] and [13]. We will modify now some definitions of LMS –introduced in section 2– in order to obtain a DMMS:

Statement 1 *In DMMS every membrane is an output membrane, except the skin membrane.*

Statement 2 Semantic domain $\mathcal{D}$ *of a membrane* M_1 *is a set over* Σ *associated to the membrane in every state of the computation. It is the set of semantemes this membrane accepts.*

Statement 3 Linguistic domain $\mathcal{E}$ *of a membrane* M_1 *is a set over* V *associated to the membrane in every state of the computation. It is the set of linguemes this membrane accepts.*

Statement 4 *A* membrane M_n, *in a DMMS, is defined in each state by means of two items: a) its semantic domain,* $\mathcal{D}$, *and b) its linguistic domain,* $\mathcal{E}$. *Thus,* $M_n = (\mathcal{D}M_n, \mathcal{E}M_n)$.

Statement 5 *The function* h *applied to a membrane* M_n *establishes a correspondence between elements of* $\mathcal{E}$ *and* $\mathcal{D}$ *associated to this membrane. For instance, the function* $h(\mathcal{D}M_n \to \mathcal{E}M_n)$ *establishes the correspondence between semantemes and linguemes belonging to the membrane* M_n. *The rules of this function, called transposition rules, have the following form:* $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \ldots, \pi \to m\}$. *Non-injective functions are allowed.*

Statement 6 *The subscript* $_i$ *attached to an element in a membrane* M_n *means that there is not a transposition rule for this element in* $h(M_n)$. *When computation finishes, every lingueme will be converted in the corresponding semanteme, following transposition rules. Elements marked with* $_i$ *are not taken into account as output of the membrane system when computation stops, this is, they disappear.*

Statement 7 *For a given skin membrane M_s, $\mathcal{D}M_s$ is* $\bigcup(\mathcal{D}M_a, .., \mathcal{D}M_n)$ $\forall\ M_a, .., M_n \in \mu$

Domains associated to each membrane can evolve. If they do it, they are called *variable domains*. The processes for changing domains are the following four:

1. *New semantemes are added to the domain.*
   For example, for $\mathcal{D}M_n = \{\alpha, \beta\}$, the rule $ADD\ \{\gamma\}\ TO\ \mathcal{D}M_n$ increases the domain, being $\mathcal{D}M_n = \{\alpha, \beta, \gamma\}$.

2. *Some semantemes are deleted from the domain.*
   For example, for $\mathcal{D}M_n = \{\alpha, \beta, \gamma\}$, the rule $DEL\ \{\beta\}\ FROM\ \mathcal{D}M_n$ decreases the domain, being $\mathcal{D}M_n = \{\alpha, \gamma\}$. When a semanteme is deleted for a domain, any rule containing it disappears.

3. *New linguemes are added to the domain.*
   For example, for $\mathcal{E}M_n = \{a, b\}$, the rule $ADD\ \{c\}\ TO\ \mathcal{E}M_n$ increases the domain, being $\mathcal{E}M_n = \{a, b, c\}$.

4. *Some linguemes are deleted from the domain.*
   For example, for $\mathcal{E}M_n = \{a, b, c\}$, the rule $DEL\ \{b\}\ FROM\ \mathcal{E}M_n$ decreases the domain, being $\mathcal{E}M_n = \{a, c\}$. When a lingueme is deleted for a domain, any rule containing it disappears.

Now, starting by the configuration given in Figure 1, we define some **rules with membranes:**

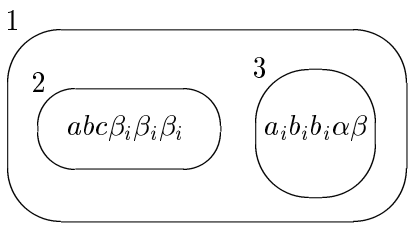

Figure 1: Basic Configuration.

1. **Deletion** is the operation by means of which a membrane M_n is dissolved and its elements go to the immediately external membrane. These elements will be accepted or rejected according to the definition of the new membrane. The rule for deleting membrane M_n is written as δM_n.

   Example 1 *Let* Π *be a membrane system with three membranes,* $[_1\ [_2\ ]_2\ [_3\ ]_3\ ]_1$, *where* $V_1 = \{a, b, c, d\}$, $V_2 = \{\alpha, \beta\}$, $\mathcal{D}M_2 = V_1$, $\mathcal{D}M_3 = V_2$, $\mathcal{D}M_1 = \mathcal{D}M_2 \cup \mathcal{D}M_3$.

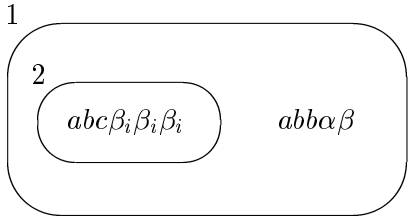

Figure 2: Deletion.

Assume that after some evolution steps the configuration reached is as shown in Figure 1. In this moment, the rule δM_3 is applied, with the result shown in Figure 2.

2. **Expansion** is the operation by means of which a membrane M_n can be expanded to other adjacent or external membranes using the rule ψM_n TO M_m, M_k. That means that membranes M_m and M_k are dissolved in M_n and their elements must be reformulated following the definition of M_n.

   Example 2 *By applying the rule ψM_3 TO M_2 to the system in the previous example, we obtain the situation in Figure 3.*

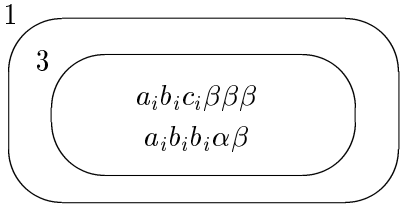

Figure 3: Expansion.

3. **Absorption** is the operation by means of which a membrane M_n disappears dissolved in another adjacent or external membrane M_m. Its elements must be reformulated according to the definition of M_m. The rule is ϕM_n IN $_m$.

   Example 3 *If we apply ϕM_3 IN M_2 to Π, the result is the system in Figure 4.*

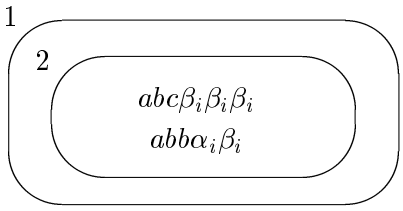

Figure 4: Absorption.

4. **Erasing** is the operation by means of which, given a membrane M_n, it can completely disappear with all its elements. The rule is χM_n.

   Example 4 *If we apply χM_3 to Π, the result is the system in Figure 5.*

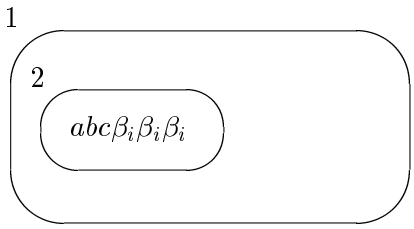

Figure 5: Erasing.

5. **Fusion** is the operation by means of which, given two membranes M_n, M_m, they joint creating M_z in such a way that $\mathcal{D}M_z = \mathcal{D}M_n \cup \mathcal{D}M_m$, $\mathcal{E}_z = \mathcal{E}_n \cup \mathcal{E}_m$. The rule is $M_m \nu M_n$.

   Example 5 *If we apply $M_2 \nu M_3$ to Π, the result is the system in Figure 6.*

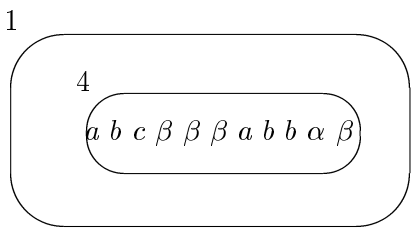

Figure 6: Fusion.

Above we have introduced rules that can be applied to membranes. Now, we refer to possible **relations among membranes**. Given two membranes M_1, M_2, we call the relation between them:

- **Nesting**, when one is inside the other. It is denoted by $M_1 \subset M_2$.
- **Sibling**, when they are adjacent or they are inside adjacent membranes and they have the same depth. It is denoted by $M_1 \approx M_2$.
- **Command**, they are not nested and they do not have the same depth. It is denoted by $M_1 \wedge M_2$.

Membranes in a system are related by what we call *communication channels*. This is, they can be communicated among them or not. There are three states of communication:

- **Connexion**: Two membranes are connected when the communication channel between them is open. It is denoted by $\odot$. A membrane is *connected* when it has the relation of connexion with every membrane in the system. A system is called *supra-connected* when every communication channel is open.
- **Isolation**: Two membranes are isolated when the communication channel between them is closed. It is denoted by $\oslash$. A membrane is *isolated* when the communication with every membrane in the system is closed. A system is called *supra-isolated* when every communication channel is closed.
- **Inhibition**: Two membranes are related by inhibition when the communication between them is closed and it cannot be opened. It is denoted by $\otimes$. A membrane is inhibited when every communication channel is closed to it, and it cannot be opened. A system is *supra-inhibited* when every membrane in it is inhibited. An inhibited system cannot work. It is not a system.

There are some properties of connexion which can be established depending on the relation between membranes:

- Two nested membranes are connected by definition.
- If a membrane is connected to a sibling membrane, then it is connected to any membrane nested to it.
- If a membrane is isolated with respect to a sibling membrane, then it is isolated from any membrane nested to it.
- If a membrane M_1 is connected to a membrane M_2, and M_2 is connected to M_3, then there is a way of communication between M_1 and M_3, even if $M_1 \oslash M_3$.

During the process of computation some specific rules can break the initial relation among membranes described above.

Example 6 *We have a membrane system with the following structure:*

$$\mu = [_S\ [_1\ [_2\ ]_2\ ]_1\ [_3\ ]_3 [_4\ ]_4\ ]_S,$$

where:

- $M_2 \subset M_1$,
- $M_1 \oslash M_3$,
- $M_1 \odot M_4$,
- $M_3 \odot M_4$.

This system is represented as shown in Figure 7.

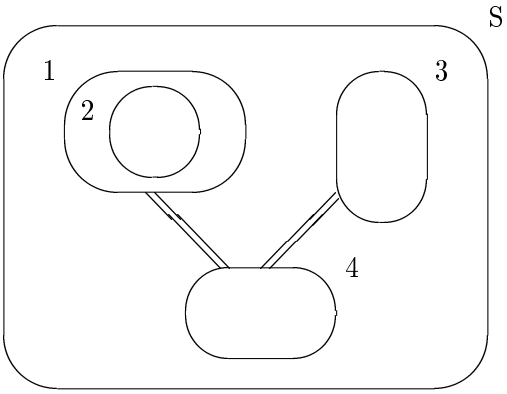

Figure 7: Example.

4 Explaining Semantic Change with DMMS

4.1 Generalities about Semantic Change

Language change can, and should, be studied at different levels. Even though no component of natural language is totally immune to change, they show a distinct level of susceptibility to change. In general, it has been said that semantics is most easily and radically affected by change than phonology, morphology and syntax, in that specific order. This is, it seems that there is less resistance to change in the semantics that in other areas of the grammar, so that meaning changes relatively quickly and easily.

According to [15], semantic change is so ubiquitous that hardly a word in the dictionary lacks earlier meanings more or less different from the present-day sense(s). However, semantic change seems to be the area of diachronic linguistics that is least well understood, perhaps because semantics has for a long time been the weak point in synchronic language study. In contrast with this, phonological change has been fairly intensively studied in the world's languages and grammatical change, even being less well studied than phonology, it is an area that is receiving a lot of attention from linguists at the present. Anyway, it is not difficult to find in the historical linguistics literature lots of observations about the kinds of semantic changes that take place in natural languages as well as about the aspects of language that allow semantic change to occur. It is precisely on these aspects in which we will concentrate on in this section. For more information about semantic change see [11, 6, 15, 1].

There seem to be different aspects of language in general, and meaning in particular, which allow semantic change to occur. Two of this aspects are the following (cfr. [11]):

1. *Polysemy*: words are typically polysemic each has various meanings or covers a whole range of shades of meaning. This flexibility is necessary since words are used in a wide variety of contexts by many different speakers, who may vary in the meaning they wish to convey. Words can lose or gain meanings

relatively easily due to this elasticity, and they do not have to lose an earlier sense to gain a new one.

2. *Arbitrariness of the linguistic sign*: they sign is bipartite, made up of a signifier and a signified. These two components are arbitrarily linked. Arbitrariness allows us to regard the signifier and the signified as essentially independent, either may therefore change with time.

Following [6], we can establish the following taxonomy of semantic evolution that considers three basic types of changes in meaning. Notice that the classification below refers to the evolution of meaning (signifier). However, according to the arbitrariness of the linguistic sign we have said that signifier and signified are independent and they may change also independently. Later we will briefly consider possible changes in the signifier, but now we are only interested on the following typology of semantic change, this is changes that affect the *signified* of a word:

- **Broadening, extension or generalization**. The term *broadening* is used to refer to a change in meaning that results in a word acquiring additional meanings to those that it originally had, while still retaining those original meanings as part of the new meaning. So, it refers to the increase of the number of contexts in which a word can be used, paradoxically reducing the amount of information conveyed about each one.

- **Narrowing, restriction or specialization**. Semantic narrowing is the exact opposite of the previous kind of change. We say that narrowing takes place when a word comes to refer to only part of the original meaning. The restriction of meaning paradoxically involves an increase in the information conveyed, since a restricted form is applicable to fewer situations, but tells more about each one.

- **Shift** occurs when a word completely loses its original meaning and acquires a new meaning. Words obviously do not jump randomly from one meaning to another when they undergo semantic shift. They may shift in smaller steps. But, as some original meanings are lost, the points of connection between intermediate semantic stages may also be lost.

The taxonomy we have just presented is the most general (and maybe useful) one about semantic change. In the literature, we can find different subtypes of each of the above three main categories of meaning changes. For example, subtypes of *generalization* are *metonymy*, *metaphorical extension* and *radiation*. A subtype of *narrowing* can be the so-called *contextual specialization*. And finally, subtypes of *shift* can be the phenomena referred to as *melioration*, *pejoration* and *semantic reversal*, etc. Since what we want to offer is a general formal framework for semantic change, we will concentrate in what follows just in the general taxonomy of meaning evolution, this is in the main three types of change: generalization, narrowing and shift.

4.2 Semantic Change from DMMS

Let us consider the above classification of semantic change and look at DMMS features introduced in section 3. Dynamic meaning membrane systems offer two powerful mechanisms that allow us to provide a membrane's explanation of the three above types of semantic change. These two mechanisms are what we have called *variable domains* and *transposition rules*.

The first type of semantic change we have referred to is the so-called **broadening, extension or generalization**. We have said that it consists on a change in meaning that results in a word acquiring additional meanings to those that it originally had, while still retaining those original meanings as part of the new meaning. This can be easily explained in DMMS by postulating the *addition of new semantemes* to the semantic domain of a membrane without adding any new lingueme. Notice that if we add new semantemes, they should be related, via $h(M_n)$, to existing linguemes. If it is the case that such linguemes, to which we associate the new semantemes, already exist in the membrane and they are already related to different semantemes, what we obtain is the extension of meaning. So, by postulating the addition of semantemes to a membrane domain we account for the gain of meaning of a word.

Formally, in DMMS, broadening is explained as follows: Given a semantic domain $\mathcal{D}M_n = \{\alpha, \beta\}$ and a linguistic domain $\mathcal{E}M_n = \{a, b\}$, apply $ADD\ \{\gamma\} TO\ \mathcal{D}M_n$. The result of this rule is the increasing of the semantic domain of M_n that changes to $\mathcal{D}M_n = \{\alpha, \beta, \gamma\}$. Now, if the linguistic domain $\mathcal{E}M_n$ does not suffer any modification, we will have to relate the added semanteme γ to an already existing lingueme, changing in this way transposition rules. The function $h(\mathcal{D}M_n \to \mathcal{E}M_n)$ establishes the correspondence between semantemes and linguemes belonging to the membrane M_n. If in the first state of computation $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \beta \to b\}$, now because of the application of $ADD\ \{\gamma\}\ TO\ \mathcal{D}M_n$, we should modify the function: $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \gamma \to a, \beta \to b\}$. See Figure 8.

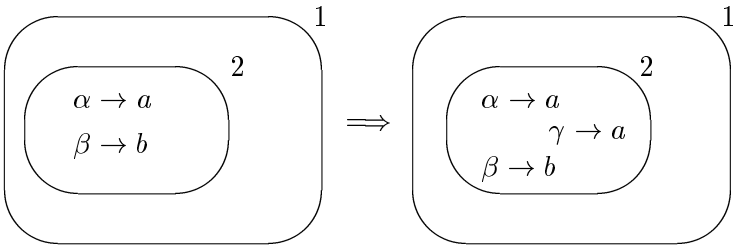

Figure 8: Broadening, Extension, Generalization.

From here, membranes easily explain examples in real life. Let us consider the following two examples:

1. The modern English *dog* derives from the earlier form *dogge*, which was originally a particularly powerful 'breed of dog' that originated in England. This in DMMS is explained as shown in Figure 9.

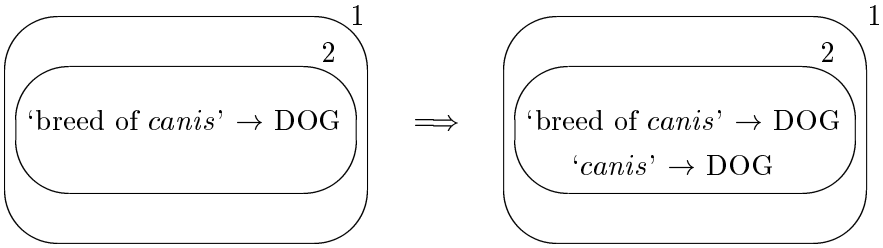

Figure 9: Example of Broadening: *dog.*

2. Let us consider the Latin word *muliere* whose meaning was 'woman'. The Spanish evolution of that Latin word –*mujer*– has acquired an additional meaning to the one that it originally had: *mujer* < *muliere* means both 'woman' and 'wife', thus we can say that an extension of meaning has taken place. This is easily explained by using the DMMS framework as shown in Figure 10.

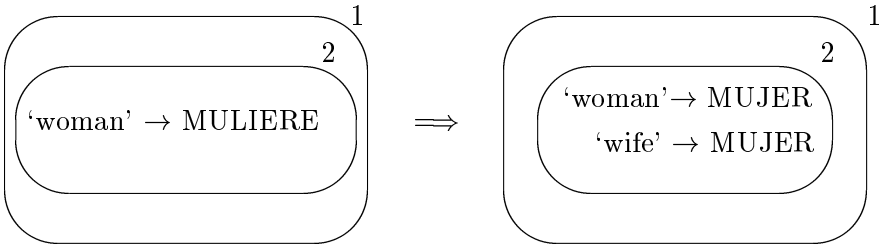

Figure 10: Example of Broadening: *muliere.*

Narrowing, restriction or specialization is the second type of semantic change we have identified in the taxonomy introduced in the above section. Semantic narrowing is the exact opposite of the previous kind of change. Here a word comes to refer to only part of its original meaning. In order to account for *restriction* of meaning in terms of DMMS, what we do is to *delete semantemes* from the domain of a membrane. Notice that using this process we can easily account for the loss of meaning of a word. If a lingueme is related to several semantemes and we delete one of those semantemes from the semantic domain of the membrane, we lose one meaning of this word. So, as it could not be otherwise, deletion of semantemes account for *restriction* of meaning.

Formally, in DMMS, narrowing is explained as follows: Given a semantic domain $\mathcal{D}M_n = \{\alpha, \beta, \gamma\}$ and a linguistic domain $\mathcal{E}M_n = \{a, b\}$, we apply the rule $DEL\ \{\gamma\}\ FROM\ \mathcal{D}M_n$. The result of this rule is the deletion of a semanteme from the semantic domain of M_n that changes to $\mathcal{D}M_n = \{\alpha, \beta\}$. When a semanteme is deleted from a domain, any rule containing it disappears. So, if in the first state of computation the function $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \gamma \to a, \beta \to b\}$, now because of the application of $DEL\ \{\gamma\}\ FROM\ \mathcal{D}M_n$, we should cancel the transposition

rule that contained the deleted semanteme, changing in this way the function to $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \beta \to b\}$. See Figure 11.

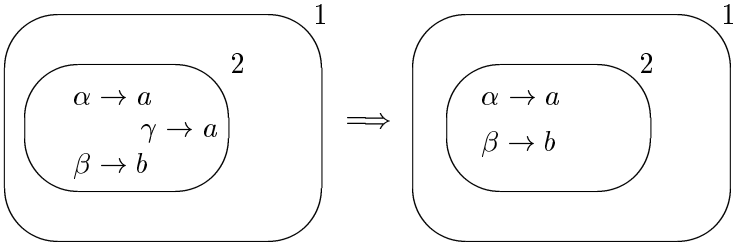

Figure 11: Narrowing, Restriction or Specialization.

This formalization helps to give account of some changes in the meaning of words like the following ones:

1. The word *hound* in English (originally pronounced *hund*) was the generic word for 'any kind of dog' at all. Over the centuries, however, the meaning of *hund* in English has become restricted to just those dogs which are used to chase game in the hunt, such as beagles. This restriction of meaning is formalized as shown in Figure 12.

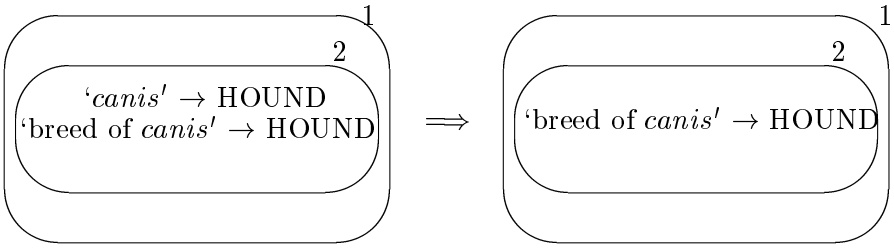

Figure 12: Example of Narrowing: *hound*.

2. The word *meat* originally referred to 'any kind of food' at all (and this original meaning is still reflected in the word *sweetmeats*) though now it only refers to food that derives from the flesh of slaughtered animals. In fact, in modern English, *meat* came to mean 'the flesh of animals as opposed to the flesh of fish'. Figure 13 shows how to explain this restriction of meaning by using DMMS.

The third and last type of semantic change is the so-called **shift**. We have an example of shift whenever a word completely loses its original meaning and acquires a new meaning. So, in order to account for this by using DMMS we have to center on *transposition rules*. In contrast with narrowing and extension, here we will not speak about addition or deletion of semantemes, but we have to postulate *a change in the transposition rule* that relates a given semanteme to a given lingueme. Let us consider that, at a given moment of computation, lingueme a is related to semanteme

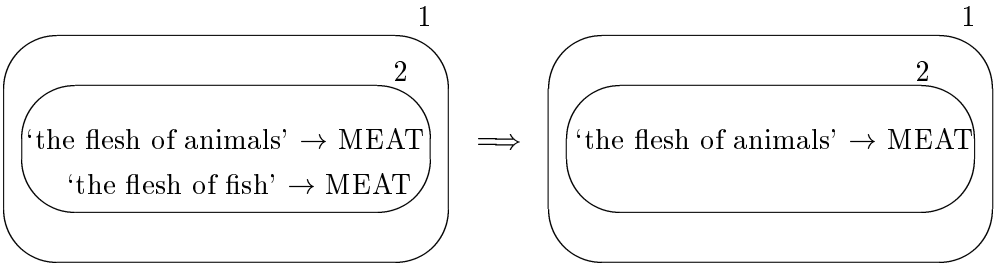

Figure 13: Example of Narrowing: *meat.*

α, we will have a case of shift if, in the next computation step, a will be related not to α anymore but to, for example, β.

Formally, in DMMS, shift is explained as follows: Consider that in the first state of computation we have a semantic domain $\mathcal{D}M_n = \{\alpha, \beta\}$, a linguistic domain $\mathcal{E}M_n = \{a, b\}$, and a function $hM_n(\mathcal{D} \to \mathcal{E}) = \{\alpha \to a, \beta \to b\}$. We will speak of shift if, in the next computation step, we will perform some modification in the transposition rules, in such a way that for a given lingueme the semanteme associated to it in the second step of computation will be different from the one it had associated in the first step. So, we have shift if there is a modification of the initial function, as for example the following one: $hM_n(\mathcal{D} \to \mathcal{E}) = \{\beta \to a, \alpha \to b\}$. See Figure 14.

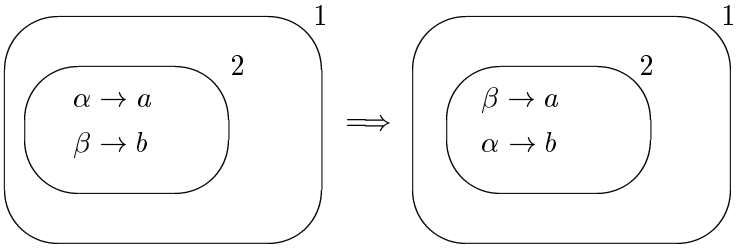

Figure 14: Shift.

Semantics provides a large number of examples of linguistic shift. An example of shift can be the history of the word *silly* in English. This word is cognate with the German word *selig* 'blessed', and it is derived from *Seele* 'soul'. The meaning of the German word represents the original meaning of the word, so there has clearly been a major semantic shift to get from the meaning 'blessed' to the meaning in modern English of 'stupid' or 'reckless'. This semantic shift is formalized in DMMS as shown in Figure 15.

From what we have said above, the proposed taxonomy of semantic change can be explain in terms of DMMS as follows:

- **Broadening:** *addition of new semantemes* to the domain of a membrane without addition of linguemes, with the obligation of relating these new semantemes to already existing linguemes (related to other semantemes).

- **Narrowing:** *deletion of semantemes* from the semantic domain of a mem-

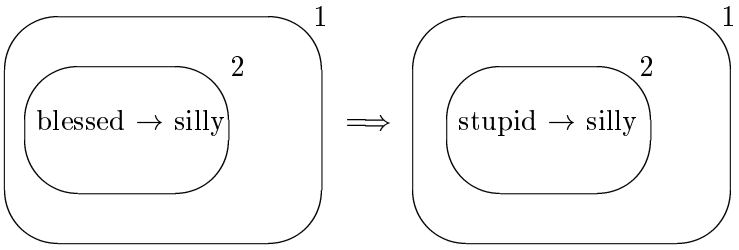

Figure 15: Example of Shift: *silly.*

brane, with the consequent deletion of every rule that contains the deleted semantemes.

- **Shift**: *a change in the transposition rule* that relates a given semanteme to a given lingueme, relating in this way the given lingueme to a different semanteme.

Therefore, by using the process of addition/deletion of semantemes we can explain *extension* and *restriction* of meaning while modification of the transposition can explain *shift* of meaning.

Up to now we have account for semantic change, this is for evolution in the signified of words (*semantemes*, in our terminology). But, not only the signified is allowed to change, in fact, in the previous section, we have said that both signifier and signified can change independently. So, in order to complete the picture of DMMS utility in language change, and even though it is not the topic of this paper, we will briefly refer to the possible DMMS' explanation of the evolution of signifiers of words (*linguemes*, in our terminology). We can use again the idea of *variable domains* and play, now, with *addition/deletion of linguemes*:

1. New linguemes can be added to a membrane domain. The addition of new linguemes can account for the introduction of a new signifier that may then be related to new or already existing meanings (semantemes). In this case $h(M_n)$ associates a new or already existing semanteme to the new introduced lingueme.

2. Some linguemes can be deleted from the domain. This process account for the elimination of a signifier whose meaning should be absorbed by another new or existing lingueme. Notice that in order to describe this fact we just need that $h(M_n)$ associates the semanteme that was associated to the lost lingueme to a different lingueme.

Above we have account separately for the change (evolution) of meaning and for the change (evolution) of signifiers. Notice that, with DMMS, we can also account for the total deletion of a word by using subscript i: all elements marked with a subscript i disappears because there is not a transposition rule for this element in $h(M_n)$ (see page 5). So every word marked will this subscript will stand for total loss of a word.

Summing up, in this section we have tried to account for semantic change by using DMMS. At the end, what we have is that DMMS can account separately for semantic change –understood in terms of addition/deletion of semantemes from a membrane domain– and for lexical (signifier) change –understood as addition/deletion of linguemes.

5 DMMS and GS: Suggestions for the Future

In the above section we have defined in terms of DMMS different types of semantic change. Note that by doing this what we have done is to answer to the question *'what?'*, this is *what does it happen when a semantic change takes place?'*. As we have seen, there are three possible answers to this question in terms of membrane systems: 1) a new semanteme is added to the membrane domain; 2) an existing semanteme is deleted from the membrane domain; 3) or there is a modification in the transposition rule that relates a given lingueme to a given semanteme. But, by just answering to the question *what?* we do not offer a complete view of language (semantic, in this case) change. There are other questions that must be answered if we like to offer a formal framework for evolution in language. Questions such as the following ones:

- *'How do linguistic changes spread through a language community?* Changes in language are interesting when they become general in a community, but obviously innovations does not originate collectively. In fact they have to be uttered for the first time by a single speaker. Some of these novelties will be imitated by other speakers, and then imitated by still others. If the process is sustained, the innovations will spread through the whole speech community. According to [11], if we recognize that there are individual idiolects, shared norms and an idealised linguistic system, we can study language change in all of these areas, and their possible interrelations: that is, an individual or group of individuals may produce a novel pronunciation or other form of speech, which contributes to variation in the speech community; this may be adopted by more speakers, and cause a change in the norms of the community; and finally, it may become the expected, or standard usage, being incorporated into a shared linguistic system of native speakers of the language.

- *'Why do particular changes spread?.* According to [6], language change can be seen as *natural*. If speakers of natural languages let things take their natural course, language will inevitably change in one way or another, given sufficient time. However there are situations in which the deliberate action of speakers can affect the future of a language. In times of rapid social, cultural and technological change, speakers of a language need to add new words to their vocabulary in order to talk about new things. So, some changes take place because a particular language must change in order to meet new demands that its speakers place upon it. As the functional needs of a language change, some aspects of the language may be lost, while others may be added. These kinds

of pressures do not generally affect the phonology, or even the grammar, but they can have drastic effects on the vocabulary.

- *'Who makes language change?* There is still a great deal of research and discussion about which social groups introduce linguistic changes. One answer seems to be that a linguistic change may enter a speech community through any social group. Members of the group with most social status, for example, tend to introduce changes into a speech community from neighbouring communities which have greater status and prestige in their eyes.

In order to account, in a formal way, for the above issues we cannot use membrane systems, but it is preferably to resort to grammar systems theory, and in particular to the so-called *cultural grammar systems*, introduced in [9] as a formal-language framework to account for cultural evolution, in general, and language evolution in particular. Cultural grammar systems –represented in Figure 16– offer a simple machinery to account for the dynamics of cultural evolution, pointing out the difference between cultural and genetic change and stressing the role of man in the evolution of culture. Taking into account that language can be seen as a part of culture, we can explain language change by using the model defined for cultural evolution.

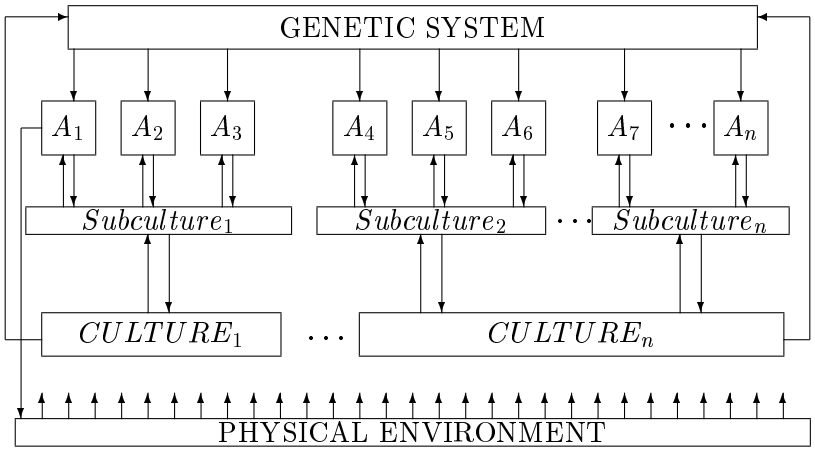

Figure 16: Cultural Grammar Systems.

Lack of space prevent us from present here the formal framework of cultural grammar systems (for formal definitions see [9]) and from describing language change by using the model. We just want to point out that with a cultural grammar system we can formally define *how* a semantic innovation spreads first from a single speaker to the most immediate linguistic community (dialect), and then to the whole linguistic community. We can capture also the idea that speakers and only speakers are *who* change languages. It remains clear in the model that it is not languages that change, but speakers who change them. And last but not least, the model offers an answer to the question *why* semantic change takes place: language changes as far as speaker's necessities change.

The only question that seems to remain unanswered in the grammar systems model is *what* happens when a semantic change takes place. And here is when enters the model we have presented in this paper. We think that there exists a complementarity between DMMS and cultural grammar systems since the former offers a model of *what* happens when a semantic change is implemented, while the latter provides explanation of the reasons, agents and procedure of the evolution of meaning. Taking into account this and considering the fact that in grammar systems we can postulate any mechanism as component of the system, we propose to define cultural grammar systems with dynamic meaning membrane systems as components in order to give a complete formal-language-model of semantic change. We are fully convinced that collaboration between both models –cultural grammar systems and DMMS- may provide a useful formalism that, due to its naturalness and simplicity, might offer interesting results in a discipline traditionally far away from any formalization as linguistic evolution is.

6 Final Remarks

One of the most challenging problems in linguistics is the possibility of dealing with meaning contexts, in order to be able to formalize one of the most important components of human language, which is the semantic module. Semantics cannot be solved in the standard theories of language because of its ambiguity and change. Moreover, the multiplicity of meanings is a constant handicap for developing computation resources applicable to linguistics.

In this paper, we have introduced a very simple model for explaining semantic change in the framework of membranes. Just a few examples have been given, related to very well known phenomena of meaning evolution. The model is based in two important ideas: a) the meaning is defined as a dynamic concept, with two components which are not joint in a consistent way, b) membranes related between them understood as different contexts for a lingueme can give account of semantic processes without involving syntactical or structural changes in it.

It is worthwhile to emphasize the simplicity of the model and the possibility of reaching a computational implementation which can help to solve some practical problems in computer programming.

Moreover, the inclusion of domains in membranes can be combined with other very fruitful concepts of standard theory of membrane systems to integrate syntax and semantics in the same process of language generation.

Finally, we suggest the integration of the model with the one provided by grammar systems, in order to give a whole explanation of the process of semantic change and language evolution.

References

[1] Aitchison, J. (1991), *Language Change: Progress or Decay?*, Cambridge University Press.

[2] Bel Enguix, G. (2003), Preliminaries about Some Possible Applications of P Systems in Linguistics, in Păun, Gh., Rozenberg, G., Salomaa, A. & Zandron, C., *Membrane Computing*, LNCS 2597, Springer, Berlin.

[3] Bel Enguix, G. & Gramatovici, R. (2004), Parsing with Active P Automata, Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., & Salomaa, A. (eds.), *Membrane Computing*, LNCS 2933, Springer, Berlin.

[4] Calude, C. & Păun, Gh. (2001), *Computing with Cells and Atoms*, Taylor and Francis, London.

[5] Croft, W. (2000), *Explaining Language Change. An Evolutionary Approach*, Longman, Singapore.

[6] Crowley, T. (1992), *An Introduction to Historical Linguistics*, Oxford University Press.

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

[8] Holmes, J. (1992), *An Introduction to Sociolinguistics*, Longman, London.

[9] Jiménez-López, M.D. (2000), *Grammar Systems: A Formal-Language-Theoretic Framework for Linguistics and Cultural Evolution*, PhD Dissertation, Rovira i Virgili University, Tarragona.

[10] Jiménez-López, M.D. (2002), *Using Grammar Systems*, GRLMC Report, Rovira i Virgili University, Tarragona.

[11] McMahon, A.M.S. (1996), *Understanding Language Change*, Cambridge University Press.

[12] Păun, Gh. (2000), Computing with Membranes, *Journal of Computer and System Sciences*, 61: 108-143.

[13] Păun, Gh. (2002), *Membrane Computing. An Introduction*, Springer, Berlin.

[14] Romaine, S. (2000), *Language in Society. An Introduction to Sociolinguistics*, Oxford University Press.

[15] Sihler, A. L. (2000), *Language History. An Introduction*, John Benjamins, Amsterdam.

[16] The P Systems Web Page: http://psystems.disco.unimib.it

[17] Wardhaugh, R. (1986), *An Introduction to Sociolinguistics*, Blackwell, Oxford.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 50 - 65, 2004.

Active Symbols in Pure Systems

Suna Bensch, Henning Bordihn

Universität Potsdam, Institut für Informatik
August-Bebel-Str. 89, 14482 Potsdam, Germany
{aydin, henning}@cs.uni-potsdam.de

Abstract

In this paper, we consider the number of (statically measured) active symbols for Lindenmayer systems without interaction and some variants thereof as well as for pure CD grammar systems, where no distinction between terminal and nonterminal symbols is made. This measure of descriptional complexity gives rise to infinite hierarchies in all cases considered here. Moreover, all the devices under consideration are compared with respect to their generative power when the number of active symbols is bounded. Finally, some closure and non-closure properties of the corresponding language families with a fixed number of active symbols are proved.

1 Introduction

In the last years the concept of active symbols was studied in several papers [5, 8, 10, 14, 15] within the framework of extended tabled Lindenmayer systems without interaction (ET0L systems). A symbols is said to be active if and only if it can be non-identically rewritten. ¿From the biological point of view active symbols can be interpreted as the maximum number of cells which are simultaneously contributing to the growing of the organism.

The authors in [2] investigate the concept of active symbols for deterministic ET0L systems (EDT0L systems) as well as for cooperating distributed grammar systems (CD grammar systems for short) working in the t-mode of derivation. Furthermore, they introduce the notion of *dynamically active symbols.*

In this paper we will only consider the *statically measured active symbols* which we will refer to as *active symbols* throughout the paper.

CD grammar systems have been introduced in [3] and have further been investigated in [4] as models of distributed problem solving. CD grammar system with context-free productions can be viewed as a generalization of context-free grammars in which the set of rules is divided into parts which are called components of the system. These components work on a common sentential form in turns according to some cooperation protocol, which determines when a component is allowed to start and to stop rewriting the sentential form. For example, in the so-called t-mode of derivation a component, once started, has to remain active as long as possible, that is, until none of its productions can be applied to the current sentential form. In

what follows, we always consider context-free CD grammar systems in the t-mode of derivation without further mentioning.

Context-free CD grammar system working in the t-mode of derivation can be considered as the sequential counterparts of ET0L systems, just having components instead of tables. Analogously, pure CD (pCD) grammar systems, where no distinction between terminal and nonterminal symbols is made, may be viewed as sequential counterparts of T0L systems. For a more detailed discussion see [1].

In this paper we investigate the number of active symbols in pure grammar formalisms, that is, we will study T0L systems and languages as well as their sequential counterparts, namely the pCD grammar systems and languages, within the framework of active symbols.

One reason why such pure grammars and systems are of interest is that there is no distinction between a sentential form and a word in the language generated. Thus, all information about the derivation process is stored in the language. This may be useful for purposes of syntax analysis. Moreover it may help to improve the understanding of the relationship between parallel and sequential rewriting mechanisms.

The paper is organized as follows. Section 2 provides the necessary definitions of the language generating devices under consideration. In section 3 we will define the notion of active symbols and show that various kinds of Lindenmayer systems without interaction as well as pCD grammar systems lead to infinite hierarchies induced by the measure of active symbols. In section 4 we will reprove and partly extend known hierarchies of some basic families of languages defined by Lindenmayer systems and show that these families of languages build the same kind of hierarchies when the number of active symbols is regarded and disregarded. Some closure and non-closure properties for the respective language families are investigated in section 5. In the conclusions we will summarize our results and state some unsolved problems.

2 Definitions and Preliminaries

We assume the reader to be familiar with basic notions in the theory of formal languages. With our notation we mainly follow [6]. In general, we have the following conventions: $\subseteq$ denotes set inclusion, while $\subset$ denotes strict set inclusion. Set difference will be denoted by $\setminus$. The set of positive integers is denoted by $\mathbb{N}$ and the cardinality of a set M is denoted by $\#M$. Let V be some alphabet, that is, a finite and non-empty set; by V^+ we denote the set of all nonempty words over V; if the empty word λ is included, then we use the notation V^*. For a word $x \in V^*$, its length is denoted by $|x|$. For any set $W \subseteq V$, $|x|_W$ is the number of occurrences of letters of W in the word x. Frequently, for singletons $\{a\}$ we simply write a. We consider two families $\mathcal{L}_1$ and $\mathcal{L}_2$ of languages to be equal if they distinguish from each other at most by the empty set, that is, if $\mathcal{L}_1 \setminus \{\emptyset\} = \mathcal{L}_2 \setminus \{\emptyset\}$. The families of regular, context-free and context-sensitive languages are denoted by $\mathcal{L}(\mathrm{REG})$, $\mathcal{L}(\mathrm{CF})$ and $\mathcal{L}(\mathrm{CS})$, respectively.

A *pure CD grammar system* (pCD grammar system for short) of degree n is an

$(n+2)$-tuple $G = (V, P_1, P_2, \ldots, P_n, S)$, where V is some alphabet, $\omega \in V^+$ is the axiom and, for $1 \leq i \leq n$, $P_i \subseteq V \times V^*$ is a finite set of pure context-free productions, which are called the components of G. A production sets P_i is called λ-free, if there are no productions of the form $A \to \lambda$ in P_i.

The t-derivation step according to component P_i is defined as follows: for $x, y \in V^*$, we write $x \overset{t}{\underset{i}{\Longrightarrow}} y$ if and only if one of the following conditions hold:

(i) there exist strings $x_0, x_1, \ldots, x_k$, $k \geq 0$, such that $x_0 = x$, $x_k = y$, $x_j \underset{i}{\Longrightarrow} x_{j+1}$, $0 \leq j \leq k-1$, and there is no z such that $y \underset{i}{\Longrightarrow} z$, or

(ii) $y = x$.

Here $x_j \underset{i}{\Longrightarrow} x_{j+1}$ denotes a direct derivation step in which the component P_i is applied, that is, $x_j = z_1 a z_2$, $x_{j+1} = z_1 v z_2$, for some $a \to v \in P_i$ and $z_1, z_2 \in V^*$. The set $\mathrm{SF}(x \overset{t}{\underset{i}{\Longrightarrow}} y)$ of the sentential forms of the t-derivation step according to P_i is (i) the set $\mathrm{SF}(x \overset{t}{\underset{i}{\Longrightarrow}} y) = \{x_0, x_1, \ldots, x_k\}$ or (ii) the set $\mathrm{SF}(x \overset{t}{\underset{i}{\Longrightarrow}} y) = \{x\}$, respectively. Note that the second case (ii) in this definition is not just a special case of (i) with $k = 0$, since there may be derivations $x \overset{*}{\underset{i}{\Longrightarrow}} y$, $y \neq x$, but all these derivations will not terminate.

A t-derivation in a pCD grammar system is a sequence of t-derivations according to arbitrary components of the system: for $x, y \in V^*$, we write $x \overset{t*}{\Longrightarrow} y$ if and only if there are strings $x_0, x_1, \ldots, x_k$, $k \geq 0$, such that $x_0 = x$, $x_k = y$, and $x_j \overset{t}{\underset{i_j}{\Longrightarrow}} x_{j+1}$, for $1 \leq i_j \leq n$, $0 \leq j \leq k-1$. The set $\mathrm{SF}(x \overset{t*}{\Longrightarrow} y)$ of its sentential forms is defined to be the union of the sets $\mathrm{SF}(x_j \overset{t}{\underset{i_j}{\Longrightarrow}} x_{j+1})$. The language $L(G)$ generated by a pCD grammar system G is the set of all sentential forms in a t-derivation in G starting with the axiom ω:

$$L(G) = \{\, w \in V^* \mid w \in \mathrm{SF}(\omega \overset{t*}{\Longrightarrow} y) \text{ for some } y \in V^* \,\}.$$

Note that the language consists of all words generated by iterated t-derivation steps and all the intermediate words appearing along these derivations. Sentential forms of derivations where the active component will not terminate remain excluded.

The family of languages generated by pCD grammar systems in the t-mode of derivation is denoted by $\mathcal{L}(\mathrm{pCD})$.

In order to clarify this definition, we repeat an example given in [1], characterizing all pCD languages over a one-letter alphabet.

Example 2.1 *(see [1, Example 2.2]) Every language over a one-letter alphabet in* $\mathcal{L}(\mathrm{pCD})$ *is either of the form* $L_{0-n} = \{a^n, a^{n-1}, a^{n-2}, \ldots, a, \lambda\}$ *or it contains exactly one nonempty word.*

A TF0L system is a triple $G = (\Sigma, H, \Omega)$, where Σ is the alphabet, H is a finite set of finite substitutions from Σ into Σ^*, and Ω is a finite, non-empty subset of Σ^+, called the *set of axioms of* G. A substitution h in H is called a table of G. If H

only contains homomorphisms or non-erasing substitutions, the TF0L system is said to be *deterministic* or *propagating* and is referred to as DTF0L or PTF0L system, respectively. If for some TF0L system $\#H = 1$, then the TF0L system is called an F0L system and if for some TF0L system $\#\Omega = 1$, then the TF0L system is called a T0L system.

If $v \in h(a)$, $a \in \Sigma$, the we say that $a \to v$ is a production in the table h. For x and y in Σ^*, we write $x \underset{h}{\Longrightarrow} y$ for some $h \in H$ if and only if $y \in h(x)$. Hence, subscript h refers to the table which is used. Let $x \overset{*}{\underset{h}{\Longrightarrow}} y$ denote the reflexive and transitive closure of the relation $x \underset{h}{\Longrightarrow} y$.

The language generated by G is defined as

$$L(G) = \{\, w \in \Sigma^* \mid \exists \omega \in \Omega, \omega \underset{h_{i_1}}{\Longrightarrow} w_1 \underset{h_{i_2}}{\Longrightarrow} \cdots \underset{h_{i_m}}{\Longrightarrow} w_m = w \text{ for some } m \geq 0 \text{ and } h_{i_j} \in H \text{ with } 1 \leq j \leq m \,\}.$$

Any combination of the denotations D, P, T and F leads to various classes of languages. In what follows, we will consider all the families of languages $\mathcal{L}(X)$ with $X \in \{\mathrm{P}, \mathrm{D}, \mathrm{PD}, \lambda\}\{\mathrm{T}, \lambda\}\{\mathrm{F}, \lambda\}\{0\mathrm{L}\}$. The set $\{\mathrm{P}, \mathrm{D}, \mathrm{PD}, \lambda\}\{\mathrm{T}, \lambda\}\{\mathrm{F}, \lambda\}\{0\mathrm{L}\}$ will be denoted by $\mathcal{M}$ for better readability.

Figure 1 and Figure 2 show some known hierarchies of these language families (for proofs see [7, 9, 11, 12, 13]). In these figures we write X instead of $\mathcal{L}(X)$ in order to obtain a better appearance of the figures; moreover an arrow denotes strict inclusion of the lower language family in the upper one, and if two families are not connected, then they are incomparable.

In [1], the language family $\mathcal{L}(\mathrm{pCD})$ is located in a part of this hierarchy, proving the strict inclusion $\mathcal{L}(\mathrm{pCD}) \subset \mathcal{L}(\mathrm{CS})$ and showing the incomparability of $\mathcal{L}(\mathrm{pCD})$ with $\mathcal{L}(\mathrm{CF})$ as well as with each $\mathcal{L}(X)$, $X \in \{\mathrm{P,D,PD},\lambda\}\{\mathrm{T},\lambda\}\{0\mathrm{L}\}$.

Moreover, by definition and [9, 12] the hierarchy in Figure 2 holds.

3 Active Symbols as Connected Measure of Syntactical Complexity

First, we provide the formal definition of the number of active symbols for TF0L systems and languages.

Definition 3.1 Let $G = (\Sigma, H, \omega)$ be a TF0L system. We define the number of active symbols in a table $h \in H$ by

$$as(h) = \#\{\, a \mid a \to w \in h \text{ with } a \neq w \,\}.$$

For G we set

$$as(G) = \max\{\, as(h) \mid h \in H \,\},$$

and for a language L in $\mathcal{L}(\mathrm{TF0L})$, we define

$$as_{\mathrm{T0L}}(L) = \min\{\, as(G) \mid G \text{ is a TF0L system and } L = L(G) \,\}.$$

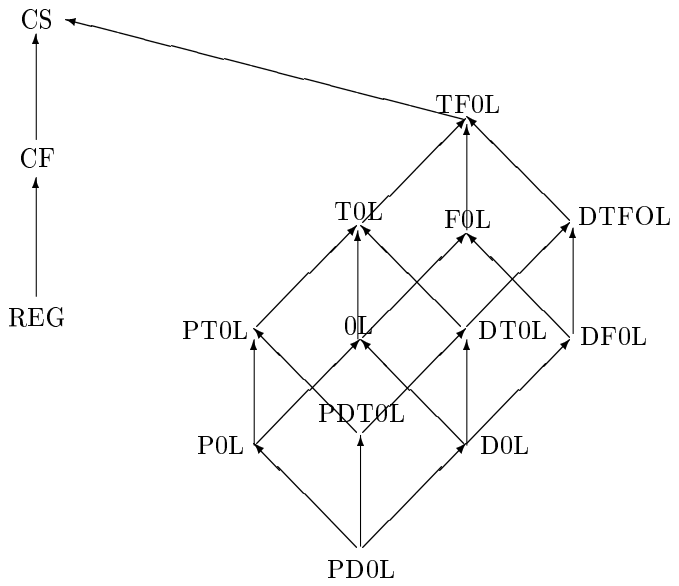

Figure 1: Preliminary hierarchy results (1)

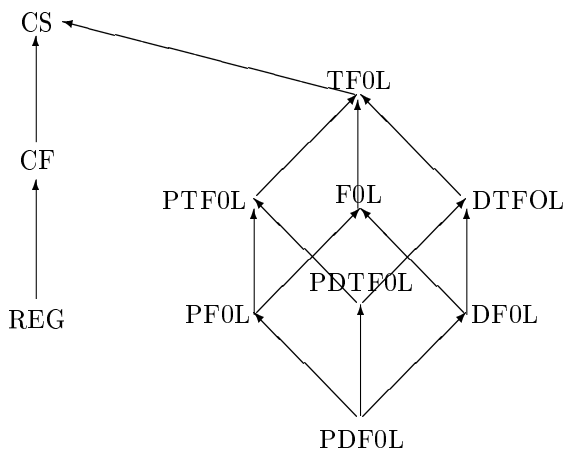

Figure 2: Preliminary hierarchy results (2)

For $X \in \mathcal{M}$ the notion $as_X(L)$ for a language L in $\mathcal{L}(X)$ is defined analogously to Definition 3.1.

In the case of pCD grammar system or CD grammar system, where components are used instead of tables, the number of the active symbols is defined analogously.

For $n \geq 0$ and $X \in \mathcal{M} \cup \{\text{pCD}\}$, let

$$\mathcal{L}(X, n) = \{\, L \in \mathcal{L}(X) \mid as_X(L) \leq n \,\}$$

be the family of languages which can be generated by some X system with at most n active symbols. By definition, $\mathcal{L}(X, n) \subseteq \text{Ł}(X, n+1)$ holds.

Moreover, the following lemmata hold by definition.

Lemma 3.1 *(i) Let $X \in \{\text{P,D,PD,}\lambda\}\{T, \lambda\}\{\text{0L}\} \cup \{\text{pCD}\}$. For any language L, $L \in \mathcal{L}(X)$, we have*

$$as_X(L) = 0 \text{ iff } \#(L) \leq 1.$$

(ii) Let $X \in \{\text{P,D,PD,}\lambda\}\{T, \lambda\}\{\text{F0L}\}$. For any language L, $L \in \mathcal{L}(X)$ we have

$$as_X(L) = 0 \text{ iff } L \text{ is finite (or empty).}$$

Lemma 3.2 *Let L be a language over a single-letter alphabet. If $L \in \mathcal{L}(X)$ with $X \in \mathcal{M}$, then $L \in \mathcal{L}(X, 1)$.*

Next, we are going to show that the number of active symbols is a connected measure of syntactical complexity with respect to all systems under consideration, that is, this measure induces infinite hierarchies of language families for all the system types considered here. More precisely, the following two theorems hold.

Theorem 3.3 *Let $X \in \mathcal{M}$. For every $n \geq 0$, there exists a language L in $\mathcal{L}(X)$ such that $as_X(L) = n$.*

Proof. The statement has been proved for $n = 0$ by Lemma 3.1. Let $n \geq 1$ and set $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$.

The language

$$L_n = \{\, a_1^{2^i} a_2^{2^i} \ldots a_n^{2^i} \mid i \geq 0 \,\}$$

is generated by the PD0L system with n active symbols

$$G_n = (\Sigma_n, \{\, a_i \to a_i^2 \mid 1 \leq i \leq n \,\}, a_1 a_2 \ldots a_n).$$

Hence, $as_X(L_n) \leq n$, for all $X \in \mathcal{M}$.

Let us assume that there is an TF0L system $G'_n = (\Sigma_n, h_1, h_2, \ldots, h_k, \omega_n)$ with $L(G'_n) = L_n$ and $as(G'_n) < n$. Then for each i, $1 \leq i \leq k$, there is at least one symbol $a_j \in \Sigma_n$ such that $h_i(a_j) = \{a_j\}$. Due to the structure of the words in L_n, any derivation step according to G'_n is of the form $u \underset{h_i}{\Longrightarrow} v$, for some $u = a_1^{2^l} a_2^{2^l} \ldots a_n^{2^l}$, $v = a_1^{2^m} a_2^{2^m} \ldots a_n^{2^m}$. Since a_j is inactive in h_i, $m = l$ has to hold. Therefore, $L(G'_n) = \{\omega_n\}$, and this contradicts $L(G'_n) = L_n$.

In conclusion, $as_X(L_n) = n$, for all $X \in \mathcal{M}$. □

Theorem 3.4 *For every $n \geq 0$, there exists a language L in $\mathcal{L}(\mathrm{pCD})$ such that $as_{\mathrm{pCD}}(L) = n$.*

Proof. For $n = 0$ the statement has been proved by Lemma 3.1. Consider the pCD grammar system of degree 2,

$$G_n = (\Sigma_n \cup \Delta_n, P_{n,1}, P_{n,2}, \omega_n), \text{ where}$$

$\Sigma_n = \{a_1, a_2, \ldots, a_n\}$, $\Delta_n = \{b_1, b_2, \ldots, b_n\}$, $\omega_n = a_1^3 a_2^3 \ldots a_n^3$, and

$$P_{n,1} = \{\, a_i \to b_i^3 \mid 1 \leq i \leq n \,\},$$
$$P_{n,2} = \{\, b_i \to a_i \mid 1 \leq i \leq n \,\}.$$

Starting off with a word $a_1^{3^i} a_2^{3^i} \ldots a_n^{3^i}$, $i \geq 1$, only $P_{n,1}$ is applicable, yielding the word $b_1^{3^{i+1}} b_2^{3^{i+1}} \ldots b_n^{3^{i+1}}$ in the t-mode of derivation.

Since the a_i's are sequentially replaced in arbitrary order, all the sentential forms in $K_{n,i} = \{u_1 u_2 \ldots u_n \mid u_j \in \{a_j, b_j^3\}^{3^i}\}$ can be obtained during all possible t-mode derivation steps starting off with $a_1^{3^i} a_2^{3^i} \ldots a_n^{3^i}$.

Next, only $P_{n,2}$ can be applied to $b_1^{3^{i+1}} b_2^{3^{i+1}} \ldots b_n^{3^{i+1}}$, where the intermediate sentential forms of all possible t-mode derivation steps build the set

$$M_{n,i} = \{\, v_1 v_2 \ldots v_n \mid v_j \in \{a_j, b_j\}^{3^{i+1}} \}.$$

Hence, $L_n = L_t(G_n) = \bigcup_{i \geq 0} (K_{n,i} \cup M_{n,i})$.

Since G_n has n active symbols, we have $as_{\mathrm{pCD}}(L_n) \leq n$.

Let $G'_n = (\Sigma_n \cup \Delta_n, P_{n,1}, P_{n,2}, \ldots, P_{n,k}, \omega'_n)$ be a pCD grammar system generating L_n in t-mode of derivation with $as(G'_n) < n$. Since for any two words w and w' of L with $|w| \neq |w'|$, we have $||w| - |w'|| \geq 2$, G' is λ-free.

Thus, the shortest word in L is the axiom, that is, $\omega'_n = a_1^3 a_2^3 \ldots a_n^3$. Let $P_{n,l}$ be a component which can successfully be applied to the axiom, that is, there is some u, $u \neq \omega'_n$, such that a derivation $\omega'_n \underset{n,l}{\overset{t}{\Longrightarrow}} u$ is possible according to G'_n.

If $a_i \to y \in P_{n,l}$, for some $1 \leq i \leq n$ and $y \in (\Sigma_n \cup \Delta_n)^*$, then this production can be applied to each occurrence of a_i in ω'_n. Therefore, $y \in \{a_i, b_i^3\}$ has to hold. Analogously, if $b_i \to y \in P_{n,l}$, then $y \in \{a_i, b_i\}$ has to hold. Since $as(P_{n,l}) < n$, there is at least one j, $1 \leq j \leq n$, such that a_j is not active, i.e., there is no production replacing a_j in $P_{n,l}$. Note that $\omega'_n \underset{n,l}{\overset{t}{\Longrightarrow}} u$ is a t-mode derivation step, where the presence of the production $a_j \to a_j$ would be blocking this derivation. In conclusion, $u = u_1 u_2 \ldots u_n$ with $u_i = \{a_i, b_i^3\}^3$, for $1 \leq i \leq n$, where $u_j = a_j^3$ holds.

Therefore, the word $x = a_1^2 b_1^3 a_2^2 b_2^3 \ldots a_n^2 b_n^3$ does not appear as intermediate sentential form during $\omega'_n \underset{n,l}{\overset{t}{\Longrightarrow}} u$.

On the other hand, because of $u \neq \omega'_n$, there is at least one j, $1 \leq j \leq n$, such that $u_j = b_j^9$.

Therefore the word x cannot be obtained during further derivations, since G'_n is λ-free.

Hence, $x \notin L_t(G'_n)$, contradicting $L_t(G'_n) = L_n$. □

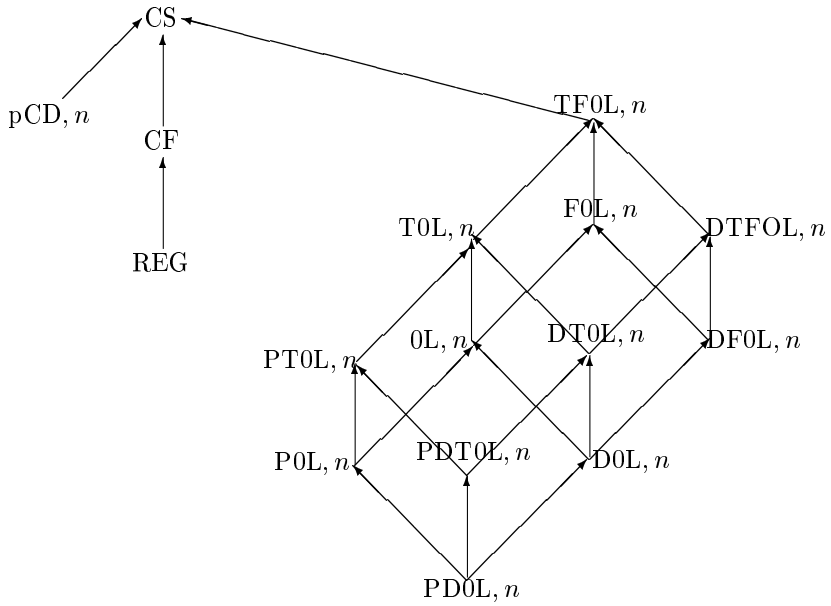

Figure 3: Hierarchy of language families with bounded number of active symbols (1)

4 Hierarchies Induced by a Bounded Number of Active Symbols

In this section we will extend the known hierarchies presented in section 2 involving the family $\mathcal{L}(\text{pCD})$. We compare the generative power of the mechanisms when the number of active symbols is taken into consideration. We are going to prove that the same hierarchical relationships are obtained both when regarding and disregarding this syntactical measure. More precisely Theorems 4.1 and 4.2 hold.

In Figure 3 and Figure 4 which are given below we write X, n instead of $\mathcal{L}(X, n)$, for $X \in \mathcal{M} \cup \{\text{pCD}\}$ (or we write REG, CF, CS instead of $\mathcal{L}(\text{REG})$, $\mathcal{L}(\text{CF})$, $\mathcal{L}(\text{CS})$, respectively) in order to obtain a better appearance of the figures; moreover an arrow denotes strict inclusion of the lower language family in the upper one, and if two families are not connected, then they are incomparable.

Theorem 4.1 *For any integer $n \geq 1$, the hierarchy presented in Figure 3 holds.*

Proof. Let $n \geq 1$. The strict inclusion $\mathcal{L}(\text{pCD}, n) \subset \text{Ł}(\text{CS})$ follows from the fact $\mathcal{L}(\text{pCD}) \subset \mathcal{L}(\text{ET0L})$ which has been shown in [1, Theorem 3.6]. The reader may readily verify that it is sufficient to show the following 10 facts.
i) Since every finite language is in $\mathcal{L}(\text{DF0L}, 0)$, the language

$$L_1 = \{a^2, ab, ba, b^2, ab^2, b^2a, b^4\}$$

is contained in $\mathcal{L}(\text{DF0L}, n)$, but L_1 is not a T0L language (see [1, Theorem 3.5]). Hence, $L_1 \in \mathcal{L}(\text{DF0L}, n) \setminus \mathcal{L}(\text{T0L}, n)$ holds.
ii) Consider the P0L language

$$L_2 = \{a\}^+$$

which can be generated by a P0L system with one active symbol, but L_2 is not a DTFOL language (see [12]). Hence, $L_2 \in \mathcal{L}(\text{P0L}, n) \setminus \mathcal{L}(\text{DTFOL}, n)$ holds.
iii) The PDT0L language

$$L_3 = \{\, \$w\$w\$w \mid w \in \{a,b\}^* \,\}$$

can be generated by the PDT0L system with one active symbol

$$G = (\{a,b,\$\}, \{\{\$ \to \$a, a \to a, b \to b\}, \{\$ \to \$b, a \to a, b \to b\}\}, \$\$\$)\,,$$

but L_3 is not an E0L language (see [13, Exercise IV.1.2]). Since every F0L language is an E0L language [7], $L_3 \in \mathcal{L}(\text{PDT0L}, n) \setminus \mathcal{L}(\text{F0L}, n)$ holds.
iv) The D0L language

$$L_4 = \{a, ab\}$$

can be generated by the D0L system $(\{a,b\}, \{a \to a, b \to \lambda\}, ab)$ having only one active symbol, but L_4 is not a PT0L language. Assume the contrary, that is, there exists a PT0L system generating L_4. Since the shortest word in L_4 is the axiom, $\omega = a$ has to hold. Then there exists at least one table h with $a \to ab \in h$ and a derivation $a \underset{h}{\Longrightarrow} ab \underset{h}{\Longrightarrow} ab\beta$, where $\beta \neq \lambda$. Therefore $|ab\beta| \geq 3$ has to hold, and $ab\beta \notin L_4$. Hence, $L_4 \in \mathcal{L}(\text{D0L}, n) \setminus \mathcal{L}(\text{PT0L}, n)$ holds.
v) The PD0L language

$$L_5 = \{\, a^{2^i} \mid i \geq 0 \,\}$$

can be generated by a PD0L grammar system with one active symbol, but L_5 is not a pCD language due to Example 2.1. Hence, $L_5 \in \mathcal{L}(\text{PD0L}, n) \setminus \mathcal{L}(\text{pCD}, n)$ holds.
vi) Conversely, consider the pCD grammar with one active symbol

$$G = (\{a,b,c,d\}, \{a \to b\}, \{a \to b^2\}, \{a \to a, c \to dc, c \to d\}, da^2c)\,.$$

Since the components $\{a \to b\}$ and $\{a \to b^2\}$ are applicable to the axiom da^2c, the words b^2 and b^4 are obtained by the respective t-mode derivations, such that the set $\{\, cxd \mid x \in \{a^2, ab, ba, b^2, ab^2, b^2a, b^4\}\}$ is a subset of the language $L_6 = L(G)$. A derivation using the component $\{a \to a, c \to dc, c \to d\}$ can not terminate if an a occurs in the sentential form; thus it is applicable only to b^2 and b^4. Therefore, G generates the language

$$\begin{aligned} L_6 \;=\; & \{da^2c, dabc, dbac, db^2c, dab^2c, db^2ac, db^4c\} \\ & \cup \{\, db^2d^ic,\ db^4d^ic,\ db^2d^i,\ db^4d^i \mid i \geq 1 \,\} \end{aligned}$$

Hence, $L_6 \in \mathcal{L}(\text{pCD}, 1)$, but L_6 is not a TF0L language.

Suppose there exists a TF0L system $G = (\{a,b,c,d\}, H, \Omega)$ generating L_6. For all $h \in H$, the following properties hold.

(1) If $d \to \delta$ in h, then $|\delta|_{\{a,b,c\}} = 0$ since h can be applied to db^4d^4 and no words in L_6 contain more than two a, more than two c, or more than four b. Furthermore, since all words in L_6 have a prefix da or db, $|\delta| = |\delta|_d \leq 1$, hence $\delta \in \{\lambda, d\}$ has to hold.

(2) If $b \to \beta$ is a rule in h, then $|\beta|_{\{a,c\}} = 0$, since $x\beta^4 y \in h(db^4c)$ for some $x, y \in \{a, b, c, d\}^*$, but there are no words in L_6 containing at least four occurrences of a or four occurrences of c. Analogously, $|\beta|_b \leq 1$ has to hold. If $|\beta|_b = 1$ and $|\beta|_d > 0$, then $x\beta^4 y \in h(db^4c)$ contains more than one subwords db, hence it is not in L_7. Thus, $\beta \in \{b\} \cup \{\, d^i \mid i \geq 0 \,\}$ has to hold. By item (1), $h(d) \in \{\lambda, d\}$. Hence, if $\beta \in d^+$, then a word with prefix d^4 is contained in $h(db^4d)$, a contradiction to the structure of the words in L_6. Therefore, $\beta \in \{\lambda, b\}$.

(3) Because of items (1) and (2) we can argue as follows: since $\delta\beta^2\delta \in h(db^2d)$, neither β nor δ can be the empty word. Hence, $b \to b$ and $d \to d$ are the only rules for b and d, respectively.

(4) Let $a \to \alpha$ be a rule in h, then $da\alpha b^2$ is prefix of all words in $h(dab^2c)$. Therefore, $\alpha \in \{\lambda, a, b^2\}$ has to hold. Since h is applicable to db^4c, $h(c)$ must be taken from d^*c or from d^*. Thus, $\alpha = \lambda$ implies $h(da^2c) \subseteq \{c, d\}^+$, a contradiction. On the other hand, $\alpha = b^2$ implies $|x|_b = 3$ for all $x \in h(dabc)$, which is a contradiction, too. Hence, $\alpha = a$.

(5) According to the above arguments, all words $h(da^2c)$ have the prefix da^2. Hence, $c \to c$ is the only rule for c in h.

In conclusion, each table h in H is the identity, and $L(G)$ is finite. This contradicts the assumption $L(G) = L_6$.

In conclusion, $L_6 \in \mathcal{L}(\text{pCD}, n) \setminus \mathcal{L}(\text{TF0L}, n)$ holds.

vii) The regular language

$$L_7 = \{\, a^i b \mid i \geq 1 \,\} \cup \{\, ab^i \mid i \geq 1 \,\}$$

is not a TF0L language (see [12, Lemma 5]). Hence, $L_7 \in \mathcal{L}(\text{REG}) \setminus \mathcal{L}(\text{TF0L}, n)$ holds.

viii) The regular language $L_8 = \{a, a^2\}$ is no pCD language due to Example 2.1, hence $L_8 \in \mathcal{L}(\text{REG}) \setminus \mathcal{L}(\text{pCD}, n)$.

ix) The language $L_5 = \{\, a^{2^i} \mid i \geq 0 \,\} \in \mathcal{L}(\text{PD0L}, 1)$ is not a CF language. Hence $L_5 \in \mathcal{L}(\text{PD0L}, n) \setminus \mathcal{L}(\text{CF})$ holds.

x) The language $L_t(G_1)$ of the proof of Theorem 3.4 is contained in $\mathcal{L}(\text{pCD}, 1)$. Assume that it is context-free. Then, its intersection with the regular set $\{a\}^+$ is context-free as well, but $\{\, a^{3^i} \mid i \geq 0 \,\}$ is not context-free, a contradiction. In conclusion, $L_t(G_1) \in \mathcal{L}(\text{pCD}, 1) \setminus \mathcal{L}(\text{CF})$. □

Theorem 4.2 *For any integer $n \geq 3$, the hierarchy presented in Figure 4 holds.*

Proof. The statement follows from Theorem 4.1 and the following facts. Let $n \geq 3$.

i) Consider the DF0L language

$$L_9 = \{\, bc(ab^2)^{2^i} \mid i \geq 0 \,\}$$

which can be generated by the DF0L system with 3 active symbols

$$G = (\{a, b, c\}, \{a \to ab^2ab^2, b \to \lambda, c \to bc\}, bcab^2)\,,$$

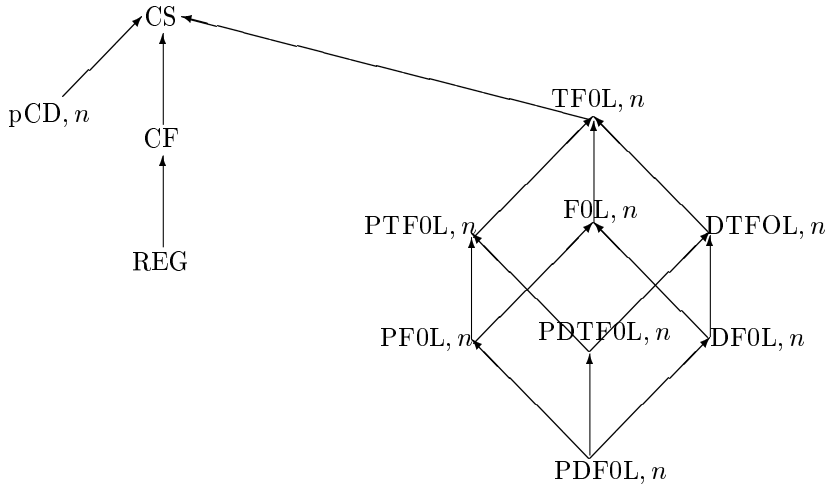

Figure 4: Hierarchy of language families with bounded number of active symbols (2)

but L_9 is not a PTF0L language (see [9, Theorem 12.1]).

Hence, $L_9 \in \mathcal{L}(\text{DF0L}, n) \setminus \mathcal{L}(\text{PTF0L}, n)$ holds.

ii) The language $L_2 = \{a\}^+ \in \mathcal{L}(\text{PF0L}, 1)$ is not a DTFOL language (see [12]). Hence, $L_2 \in \mathcal{L}(\text{PF0L}, n) \setminus \mathcal{L}(\text{DTFOL}, n)$ holds.

iii) The language $L_3 = \{\, \$w\$w\$w \mid w \in \{a, b\}^* \,\} \in \mathcal{L}(\text{PDTF0L}, 1)$ is not an F0L language (see [13, Exercise IV.1.2]). Hence, $L_3 \in \mathcal{L}(\text{PDTF0L}, n) \setminus \mathcal{L}(\text{F0L}, n)$ holds.

□

5 Closure Properties

In this section, we investigate whether or not certain operations on languages lead out of the families of languages generated by some pure system with a bounded number of active symbols. For the definitions of the considered operations we refer to any standard text book on formal languages or to [6].

Due to Lemma 3.1, we immediately obtain the following result.

Theorem 5.1 *(i) For $X \in \{\text{P,D,PD,}\lambda\}\{\text{T,}\lambda\}\{\text{0L}\} \cup \{\text{pCD}\}$, each of the families of languages $\mathcal{L}(X, 0)$ is closed with respect to product, intersection, homomorphism, and intersection with a regular set, but it is not closed with respect to union, complement, Kleene star, Kleene plus, and inverse homomorphism.*

(ii) For $X \in \{\text{P,D,PD,}\lambda\}\{\text{T,}\lambda\}\{\text{F0L}\}$, each of the families of languages $\mathcal{L}(X, 0)$ is closed with respect to union, intersection, product, homomorphism, and intersection with a regular set, but it is not closed with respect to complement, Kleene star, Kleene plus, inverse homomorphism.

Apart from this, we obtain only non-closure properties.

Theorem 5.2 *For any integer $n \geq 1$, the family of languages $\mathcal{L}(\mathrm{pCD}, n)$ is not closed with respect to union, intersection, complement, product, Kleene star, Kleene plus, intersection with a regular set, homomorphism, inverse homomorphism.*

Proof. Let $n \geq 1$.

Union.
Consider the languages $\{a\}$ and $\{a^3\}$ which are both in $\mathcal{L}(\mathrm{pCD}, n)$, but their union $\{a, a^3\}$ is not in $\mathcal{L}(\mathrm{pCD})$, thus it is not in $\mathcal{L}(\mathrm{pCD}, n)$.

Complement. The language $\{a\}^* \setminus \{a\}$ is not contained in $\mathcal{L}(\mathrm{pCD})$, hence not in $\mathcal{L}(\mathrm{pCD}, n)$.

Kleene star and Kleene plus.
The languages $\{a\}^*$ and $\{a\}^+$ are not contained in $\mathcal{L}(\mathrm{pCD}, n)$.

Intersection with a regular set.
The pCD grammar system $G = (\{a, b\}, \{a \to a^2, a \to b\}, a)$ generates the language $\{a, b\}^+$. The intersection $\{a, b\}^+ \cap \{a\}^+ = \{a\}^+$ is not in $\mathcal{L}(\mathrm{pCD}, n)$.

Intersection.
The languages $\{a, c\}^+$ and $\{a, b\}^+$ are both in $\mathcal{L}(\mathrm{pCD}, 1)$, but their intersection $\{a, c\}^+ \cap \{a, b\}^+ = \{a\}^+$ is not contained in $\mathcal{L}(\mathrm{pCD}, n)$.

Product.
Consider the languages $\{\lambda, a\}$ and $\{a^2\}$ which are both in $\mathcal{L}(\mathrm{pCD}, 1)$ (see Example 2.1). Their product $\{\lambda, a\}\{a^2\} = \{a^2, a^3\}$ is not in $\mathcal{L}(\mathrm{pCD})$, hence not in $\mathcal{L}(\mathrm{pCD}, n)$.

Homomorphism.
The pCD grammar system $G = (\{a, b\}, \{a \to b\}, a)$, generates the language $\{a, b\}$, hence it is contained in $\mathcal{L}(\mathrm{pCD}, 1)$.

Let $\phi : \{a, b\}^* \to \{a, b\}^*$ be a homomorphism such that $\phi(a) = a$ and $\phi(b) = a^3$, then $\phi\{a, b\} = \{a, a^3\}$, which is not contained in $\mathcal{L}(\mathrm{pCD}, n)$ (see Example 2.1).

Inverse homomorphism.
Consider the language $\{\lambda, a\} \in \mathcal{L}(\mathrm{pCD}, 1)$ and let $\phi : \{a\}^* \to \{a\}^*$ a homomorphism such that $\phi(a) = \lambda$. Then $h^{-1}(L) = \{a\}^*$ which is not in $\mathcal{L}(\mathrm{pCD}, n)$. □

Theorem 5.3 *(i) For any integer $n \geq 1$, the following statements hold.*

- *(a) Each of the families $\mathcal{L}(X, n)$ with $X \in \mathcal{M}$ is not closed with respect to union, complement, product, homomorphism, and intersection with a regular set.*
- *(b) Each of the families $\mathcal{L}(X, n)$ with $X \in \mathcal{M} \setminus \{\mathrm{PD0L}, \mathrm{D0L}\}$ is not closed with respect to intersection.*
- *(c) Each of the families $\mathcal{L}(X, n)$ with $X \in \mathcal{M}$ is not closed with respect to Kleene star and Kleene plus.*
- *(d) Each of the families of languages $\mathcal{L}(X, n)$ with $X \in \mathcal{M} \setminus \{\mathrm{PF0L}, \mathrm{F0L}, \mathrm{PTF0L}, \mathrm{TF0L}\}$ is not closed with respect to inverse homomorphisms.*

(ii) For any integer $n \geq 2$, the following statements hold.

(a) *The families* $\mathcal{L}(\mathrm{D0L})$ *and* $\mathcal{L}(\mathrm{PD0L})$ *are not closed with respect to intersection.*

(b) *Each of the families* $\mathcal{L}(X, n)$ *with* $X \in \{\mathrm{P},\lambda\}\{\mathrm{T},\lambda\}\{\mathrm{F},\lambda\}\{0\mathrm{L}\}$ *is not closed with respect to Kleene star and Kleene plus.*

(c) *Each of the families* $\mathcal{L}(X, n)$ *with* $X \in \{\mathrm{PF0L}, \mathrm{F0L}, \mathrm{PTF0L}, \mathrm{TF0L}\}$ *is not closed with respect to inverse homomorphisms.*

Proof.
(i) Let $n \geq 1$.

Union.
Consider the languages $\{a^3\}$ and $\{\, a^{2^i} \mid i \geq 1 \,\}$ which are both in $\mathcal{L}(\mathrm{PD0L}, n)$, but their union $\{a^3\} \cup \{\, a^{2^i} \mid i \geq 1 \,\}$ is not in $\mathcal{L}(\mathrm{TF0L})$ (see [12, Theorem 2]), thus it is not in $\mathcal{L}(\mathrm{TF0L}, n)$.

Complement.
The language $\{\, a^{2^i} \mid i \geq 1 \,\}$ is in $\mathcal{L}(\mathrm{PD0L}, n)$, but $\{a\}^* \setminus \{\, a^{2^i} \mid i \geq 1 \,\}$ is not in $\mathcal{L}(\mathrm{TF0L})$ (see [12, Theorem 2]) and therefore not contained in $\mathcal{L}(\mathrm{TF0L}, n)$.

Intersection.
Consider the P0L systems $G_1 = (\{a, b, c\}, \{a \to a, b \to b, c \to acb, c \to ab\}, c)$ and $G_2 = (\{a, b, d\}, \{a \to a, b \to b, d \to adb, d \to ab\}, d)$.

Then, $L(G_1) \cap L(G_2) = \{\, a^i b^i \mid i \geq 1 \,\}$.

Assume that this language is generated by some TF0L system G. If $a \to \alpha$ is a rule in some table of G, then $\alpha \in a^*$ has to hold, and for any rule $b \to \beta$, $\beta \in b^*$ has to hold. If there are two productions of these forms in one table with $|\alpha| \neq |\beta|$, then a string of the form $a^j b^k$ with $j \neq k$ can be obtained from ab. In conclusion, G is deterministic. Since $L(G)$ is infinite, $|\alpha| = |\beta| > 0$ in each table of G. But this leads to an exponential progression, a contradiction to our assumption. Hence, $\{\, a^i b^i \mid i \geq 1 \,\} \notin \mathcal{L}(\mathrm{TF0L})$. This shows the non-closure for all propagating classes.

Clearly, $L(G_1)$ and $L(G_2)$ are in $\mathcal{L}(\mathrm{DT0L}, 1)$, as well (just divide the productions appropriately into two tables). Thus, the non-closure for all tabled classes is proved.

Intersection with a regular set.
Use the language $L(G_1)$ of the proof for the non-closure under intersection (see *iii)*), and the regular set a^*b^*. This proves the non-closure for all propagating or tabled classes.

In order to complete the argument, consider the language $\{\, a^{2^i} \mid i \geq 0 \,\} \in \mathcal{L}(\mathrm{PD0L}, 1)$, again. Its intersection with the regular set $\{a^2, a^4\}$ is no T0L language, see [11, Theorem 2].

Product.
Consider the languages $\{a\}$ and $\{\, a^{2^i} \mid i \geq 0 \,\}$. Their product $\{a\}\{\, a^{2^i} \mid i \geq 0 \,\}$ is not in $\mathcal{L}(\mathrm{TF0L})$ (see [12, Theorem 2]) and hence not in $\mathcal{L}(\mathrm{TF0L}, n)$.

Kleene star and Kleene plus.
The claim is shown using $\{a\}$ and the fact that neither $\{a\}^*$ nor $\{a\}^+$ is a DTFOL language [12].

Homomorphism.
First, consider the PD0L language $\{a^2, b^4\}$ which is generated by the PD0L system $G = (\{a, b\}, \{a \to b^2, b \to b\}, a^2)$. Hence, the language $\{a^2, b^4\}$ is in $\mathcal{L}(\mathrm{PD0L}, 1)$.

Let $\phi : \{a,b\}^* \to \{a,b\}^*$ be a homomorphism such that $\phi(a) = \phi(b) = a$, then $\phi(\{a^2, b^4\}) = \{a^2, a^4\}$ which is not a T0L language (see [11, Theorem 2]) and therefore also not in $\mathcal{L}(\text{T0L}, n)$.

Now, consider the PDF0L system $G = (\{a,b\}\{a \to a^2, b \to b\}, \{a, b^3\})$ that is generating the language $\{b^3\} \cup \{\, a^{2^i} \mid i \geq 1 \,\}$ which is therefore in $\mathcal{L}(\text{PDF0L}, n)$. Let $\phi : \{a,b\}^* \to \{a,b\}^*$ be a homomorphism such that $\phi(a) = \phi(b) = a$, then

$$\phi(\{b^3\} \cup \{\, a^{2^i} \mid i \geq 1 \,\}) = \{a^3\} \cup \{\, a^{2^i} \mid i \geq 1 \,\},$$

which is not a TF0L language (see [12, Theorem 2]) and therefore also not contained in $\mathcal{L}(\text{TF0L}, n)$.

Inverse homomorphism.
At first, $\{a^2, b^3\} = \phi^{-1}(\{c^6\})$, if ϕ is the homomorphism $\phi : \{a,b\}^* \to \{c\}^*$ with $\phi(a) = c^3$ and $\phi(b) = c^2$. On the other hand, it is proved in [11, Theorem 2] that $\{a^2, b^3\}$ is no T0L language.

Furthermore, $\{a\}^* = h^{-1}(\{a, \lambda\})$, if h is the homomorphism defined by $h(a) = \lambda$, but $\{a\}^* \notin \mathcal{L}(\text{DTFOL})$.

(ii) Now, let $n \geq 2$.

Intersection. The D0L systems $G_1 = (\{a,b,c\}, \{a \to a^2b, b \to a^2b\}, a)$ and $G_2 = (\{a,b\}, \{a \to \lambda, b \to a\}, a^2b)$ generate the languages $\{a\} \cup \{\, (a^2b)^{3^i} \mid i \geq 0 \,\}$ and $\{\lambda, a, a^2b\}$, respectively, each with two active symbols. The intersection $L(G_1) \cap L(G_2) = \{a, a^2b\}$ is known to be no D0L language (see [11, Theorem 2]).

The PD0L systems $G_1 = (\{a,b,c\}, \{a \to ab, b \to c, c \to c\}, a)$ and $G_2 = (\{a,b,d\}, \{a \to ab, b \to d, d \to d\}, a)$ generate the languages $\{\, a, abc^n \mid n \geq 0 \,\}$ and $\{\, a, abd^n \mid n \geq 0 \,\}$, respectively. The intersection $L(G_1) \cap L(G_2) = \{a, ab\}$ is no PT0L language (see iv) in the proof of Theorem 4.1).

Kleene star and Kleene plus.
The language $\{\, a^{2^i}b^{2^i} \mid i \geq 0 \,\}$ is in $\mathcal{L}(\text{PD0L}, n)$, but $\{\, a^{2^i}b^{2^i} \mid i \geq 0 \,\}^*$ is not in $\mathcal{L}(\text{TF0L})$ (see [12, Theorem 2]) and hence not in $\mathcal{L}(\text{TF0L}, n)$. The non-closure under Kleene plus is shown analogously.

Inverse homomorphism. Consider the PF0L system

$$G = (\{a,b,c,d\}, \{a \to a, b \to b, c \to ac, c \to b, d \to bd, d \to a\}, \{ac, ad\}).$$

Let $\phi : \{a,b,c,d\}^* \to \{a,b,c,d\}^*$ be the homomorphism defined by $\phi(a) = a$, $\phi(b) = b$, $\phi(c) = \phi(d) = cd$. Then, we have

$$\phi^{-1}(L(G)) = \{\, a^ib \mid i \geq 1 \,\} \cup \{\, ab^i \mid i \geq 1 \,\},$$

which is no TF0L language [12, Lemma 5]. □

6 Concluding Remarks

The number of (statically measured) active symbols, which has extensively been investigated for ET0L , EDT0L , and CD grammar systems, is considered in the present paper for the pure versions of these systems and several variants thereof. It

is shown that this measure of syntactical complexity is connected for all cases under consideration. Next, the known hierarchies of the underlying families of languages are proved to be valid also if one bounds the number of active symbols by any constant $n \geq 1$ (or $n \geq 3$ in a few cases). One should remark that the condition $n \geq 3$ is needed only in one of the numerous constructions (which is only used for proving certain incomparability results, not affecting the strictness of any inclusion). Therefore, in principle a better result than that stated in Theorem 4.2 has been shown. Finally, some closure and, mainly, non-closure properties of the considered families of languages with bounded number of active symbols are proved, where except from a few marginal cases, optimal results could be achieved.

The problems which have been left open are:

1. Is there a language in $\mathcal{L}(\mathrm{DF0L}, 1) \setminus \mathcal{L}(\mathrm{PTF0L}, 2)$?

2. Which of $\mathcal{L}(\mathrm{D0L}, 1)$ and $\mathcal{L}(\mathrm{PD0L}, 1)$ is closed under intersection?

3. Is $\mathcal{L}(X, 1)$ for nondeterministic types X of systems closed under Kleene star or plus?

4. Is $\mathcal{L}(X, 1)$ with $X \in \{\mathrm{PF0L}, \mathrm{F0L}, \mathrm{PTF0L}, \mathrm{TF0L}\}$ closed under inverse homomorphism?

In [1] also deterministic pCD grammar systems and languages have been considered. Due to the proof of Theorem 3.4 the number of active symbols induces an infinite hierarchy also in this deterministic case, more precisely

> *For every $n \geq 0$, there exists a language L generated by a deterministic* pCD *grammar system such that* $as_{\mathrm{pCD}}(L) = n$.

Unfortunately, some constructions used in the subsequent proofs make use of nondeterministic pCD grammar systems. This leads to a lot of open problems.

Moreover, the computability of the number of active symbols for given pure languages is of interest. Finally, also the dynamic interpretation of the number of active symbols could be treated in the framework of pure systems.

References

[1] H. Bordihn, E. Csuhaj-Varjú, and J. Dassow. CD grammars systems versus L systems. In G. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, pages 18–32. Gordon and Breach Publishers, 1999.

[2] H. Bordihn and M. Holzer. On the number of active symbols in L and CD grammar systems. *Journal of Automata, Languages and Combinatorics*, 6:411–426, 2001.

[3] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics (EIK)*, 26:49–63, 1990.

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

[5] J. Dassow. $L(ETOL_{[6]}) = L(ET0L)$. Unpublished note, 1981.

[6] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Akademie-Verlag, Berlin, 1989.

[7] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 1*, pages 253–328. Springer-Verlag, 1997.

[8] H.C.M. Kleijn and G. Rozenberg. A study in parallel rewriting systems. *Information and Control*, 44:134–163, 1980.

[9] M. Nielson, G. Rozenberg, A. Salomaa, and S. Skyum. Nonterminals, homomorphisms and codings in different variations of 0L systems, ii. nondeterministic systems. *Acta Informatica*, 3:357–364, 1974.

[10] Gh. Păun and J. Dassow. $L(ETOL_{[5]}) = L(ET0L)$. *Elektronische Informationsverarbeitung und Kybernetik (J. Inf. Process. Cybern. EIK)*, 18:345–348, 1982.

[11] G. Rozenberg. T0L systems and languages. *Information and Control*, 23:357–381, 1973.

[12] G. Rozenberg and K. P. Lee. Development systems with finite axiom sets, part 1: Systems without interactions. *Intern. J. Computer Math.*, 4:43–68, 1974.

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

[14] T. Yokomori, D. Wood, and K.-J. Lange. A three-restricted normal form for ET0L languages. *Information Processing Letters*, 14:97–100, 1982.

[15] T. Yokomori, D. Wood, and K.-J. Lange. Erratum at a three-restricted normal form for ET0L languages. *Information Processing Letters*, 21:53, 1985.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 66 - 77, 2004.

Population P Systems and Grammar Systems

Francesco Bernardini, Marian Gheorghe

Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
{f.bernardini,m.gheorghe}@dcs.shef.ac.uk

Abstract

The paper introduces two preliminary variants of population P systems with string-objects. The former one is inspired by CD grammar systems regarded as a population of interacting/cooperating cells that use rewriting rules to manipulate set of strings placed inside them. In this case, cells are restricted to communicate only by means of the environment and no direct communication among the cells can take place in the system. The second variant instead uses the notion of communication mediated by query symbols considered for PC grammar systems in order to define the features of bond making rules in context of population P systems with string-objects. Some preliminary results concerning the computational power of these population P systems are reported and some directions for future research are briefly discussed.

1 Introduction

Membrane computing represents a new and rapidly growing research area which is part of the natural computing paradigm. Already a monograph has been dedicated to this subject [10] and some fairly recent results can be found in [12], [8]. Membrane computing has been introduced with the aim of defining computing devices, called P systems, which abstract from the structure and the functioning of living cells [9]. A new variant of P systems, called population P systems, has been recently introduced in [1] that extends the existing notions of P systems in the following senses:

- the structure of the system is defined as an arbitrary graph rather than as a tree; each node in the graph corresponds in an one-to-one manner to a cell in the system; cells are the basic functional units of a population P systems and they are allowed to communicate alongside the edges of the graph;
- the graph defining the structure of the system can change during a computation; these changes involve both the set of nodes and the set of edges in the graph; new cells can therefore be introduced in the system and new links can be formed among these cells by altering in this way their communication capabilities.

From a biological point of view, population P systems can be considered as an abstraction for populations of of bio-units aggregated in more complex bio-entities.

Cells represent individuals in the population that are allowed to interact/cooperate according to a specific communication model. Links between the cells model the fact that, in order to communicate, these individuals need to "get in touch" somehow. Furthermore, the operations of cell differentiation, cell division and cell death are considered for population P systems as generic mechanisms to alter the nature and the number of individuals in the system [1].

Grammar systems is another active area of theoretical computer science that advocates the use of a grammatical approach to model distribution and cooperation in a computing system [3]. Grammar systems, in their basic variant called CD grammar systems, consist of a number of distinct grammars cooperating each other according to a given protocol in order to rewrite a common string shared by all the grammars in the system. The initial motivation for CD grammar systems were related to two-level grammars and to artificial intelligence issues. In this respect, CD grammar system can be seen as a model for describing autonomous agents cooperating each other in order to solve a common problem in such a ways that resembles the blackboard paradigm used in artificial intelligence [3]. Further variants of grammar system have been then proposed that introduce in the model extra features from other areas of parallel/distributed computing and/or with some biological inspiration [3], [4].

There are evident similarities between the P system model and the grammar system model: both of them define devices where computing resources are distributed among different individual components and interaction/cooperation between these components is fundamental to achieve successful and meaningful computations. In particular it is natural to think of a grammar systems as a population of interacting/cooperating grammars that work in a common shared environment. This is the approach we adopt in this paper where we investigate population P systems that use rewriting as the basic operation to manipulate sets of strings placed inside the cells in the system. All over the paper, the reader is supposed to be familiar with the basic notions of formal language theory, the notation commonly used in membrane computing and in grammar systems as well. We refer to [13], [6], [10], [3] for further details.

2 Modelling CD Grammar Systems

A CD grammar system [3] is a construct $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ where: N is a finite set of non-terminal symbols, T is a finite set of terminal symbols, P_i, for each $1 \leq i \leq n$, is a finite set of context-free rules of the form $X \to w$, with $X \in N$, $w \in (N \cup T)^*$, and $S \in N$ is the initial symbol of the system. Notice that, without loss of generality, CD grammar systems are presented here in the form that imposes the restriction of not rewriting the terminals of a component in the other ones. In the case of context-free rules, this restriction in fact does not alter the generative capacity of CD grammar systems (see Theorem 6.1 in [3]).

Each component in a CD grammar system Γ always operates according to a specific derivation mode $\delta \in \{k, \leq k, \geq k, *, t \mid k \geq 1\}$ adopted by the system Γ (see [3] for a formal definition of these derivation modes). Specifically, given two strings

$x, y \in (N \cup T)^*$ we write $x \Longrightarrow_\Gamma^\delta y$ if and only if $x \Longrightarrow_{P_i}^\delta y$, for some $1 \leq i \leq n$. Thus, the language $L_\delta(\Gamma)$ generated by the grammar system Γ using the derivation mode δ, is the language that contains all the strings $y \in T^*$ such that there exists a derivation in Γ of the form $S \Longrightarrow_\Gamma^\delta z_1 \Longrightarrow_\Gamma^\delta \ldots \Longrightarrow_\Gamma^\delta z_h \Longrightarrow_\Gamma^\delta y$, with $h \geq 0$, $z_j \in (N \cup T)^+$.

In this framework, a population P system model for systems of cooperating grammars can be defined in a straightforward manner by considering systems consisiting of a number cells with some finite sets of context-free rule and sharing a common environment; this environment is used to move the strings to be rewritten from one cell to another one and to collect the result of a computation. More precisely, an unstructured population P system is defined as a construct $\mathcal{P} = (V, T, E, C_1, C_2, \ldots, C_n)$ where: V a is finite set of symbols, $T \subseteq V$ is a finite set of terminal symbols, $E \subseteq V^*$ is a finite language assigned to the environment and, for each $1 \leq i \leq n$, $C_i = (R_i, F_i)$ with R_i a finite set of context-free rules of the form $a \to (v, out)$, for $a \in V$, $v \in V^*$, and F_i a finite set of filters of the form (a, in), for $a \in V$.

A computation in an unstructured population P system is performed by distributing to the cells, according to their respective sets of filters, the strings in the environment, where they are rewritten by means of some rules and immediately returned to the environment so that the process can be iterated. A filter $(a, in) \in F_i$, with $a \in V$, specifies that a string in the environment can enter cell i if and only if the symbol a is present inside that string; a rule $a \to (v, out)$ in R_i specifies that a string can be rewritten inside cell i by replacing the symbol a with the string v and, after that, the string has to exit cell i and be associated with the environment. The language generated by an unstructured population P system $\mathcal{P}$ is the language $L(\mathcal{P})$ that consists of all the strings in T^* that are produced inside the environment and that cannot enter the cells anymore.

Thus, by having this notion of population P systems, it is easy to verify that the following proposition holds.

Proposition 2.1. *Let Γ be a CD grammar systems with $n \geq 1$ components. It is always possible to construct an unstructured population P system $\mathcal{P}$ such that $L(\mathcal{P}) = L_1(\Gamma)$.*

On the other hand, it is not so immediate to simulate in the context of (population) P systems derivation modes other than the 1-mode. This might require the introduction of specific operations to manipulate the cells in the system or the introduction of specific control mechanisms to regulate the communication of strings from one place to another one. However, as we will see in the next section, population P systems are in general more powerful than CD grammar systems in terms of generative capacity.

3 Unstructured Population P Systems

In this section we introduce a more general notion of unstructured population P systems and we report some results concerning their computational power.

Definition 3.1. An unstructured population P system with string-objects is a construct

$$\mathcal{P} = (V, T, L, E, C_1, C_2, \ldots, C_n, R),$$

where:

1. V is a finite alphabet;
2. $T \subseteq V$ is a finite set of terminal symbols;
3. L is a finite set of labels that defines a set of possible types for the cells in the system;
4. $E \subseteq V^*$ is a finite language initially assigned to the environment;
5. $C_i = (M_i, t_i)$, for each $1 \leq i \leq n$, with $M_i \subseteq V^*$ a finite language defining the initial content of cell i, and $t_i \in L$ the initial type of cell i;
6. R is a finite set of rules of the forms:
   (a) $(\, a \to y\,)_t$, for $a \in V$, $v \in V^*$, $t \in L$ (*transformation rules* that allow a cell of type t to rewrite one of its internal strings by replacing the symbols a with the string v),
   (b) $(\, a\,)_t \to v\,(\,)_t$, for $a \in V$, $v \in V^*$, $t \in L$ (*output rules* that allow a cell of type t to rewrite one of its internal strings and move it into the environment),
   (c) $a(\,)_t \to (\, v\,)_t$, for $a \in V$, $v \in V^*$, $t \in L$ (*input rules* that allow a cell of type t to rewrite a string in the environment and add it to the content of the cell),
   (d) $(\, a\,)_t \to (\, v\,)_p$, for $a \in V$, $v \in V^*$, $t, p \in L$(*cell differentiation rules* that allow a cell of type t to rewrite one of its internal strings and change its type from t to p)
   (e) $(\, a\,)_t \to (\, v\,)_t\,(\, z\,)_t$, for $a \in V$, $v, z \in V^*$, $t \in L$ (*cell division rules* that allow cell of type t to rewrite one of its internal strings in order to produce two cells of type t, containing the same set of string as the originating one except for the string to be rewritten where the symbol a is replaced by the string v in one cell and by the string z in the other one).

This definition is obtained as a straightforward adaption to the case of string-objects of the definition given in [1] for population P systems with active cells in the case of symbols objects. In particular, communication of strings through the environment is achieved by means of a finite set of output and input rules: these rules allow a string to be rewritten inside a cell and then moved out from that cell or, alternatively, to be rewritten inside the environment and moved into a cell.

As usual, a computation in an unstructured population P system $\mathcal{P}$ is obtained by applying in a non-deterministic maximal parallel manner the rules in R to the strings in the system by starting from the initial configuration, with the further specification that each string is rewritten in a sequential way and that at most one rule of the

form (d), or (e) per each cell can be used at a time. The output of a computation is given by the set of strings in T^* that are produced inside the environment and that cannot enter the cells anymore; as usual in P systems with string-objects, we do not work with halting computations but we accept all the strings of the aforementioned form produced by any computation in $\mathcal{P}$. The language generated by $\mathcal{P}$ in this way is then denoted by $L(\mathcal{P})$. Then, we introduce the families of languages of the form $ELPP_{n,k}(op)$, with $n, k \geq 1$, and $op \subseteq \{a, b, c, d, e\}$ that represent the families of languages generated by (unstructured) population P systems where: the number of cells in a step of computation is always less than or equal to n, the cardinality of the set of possible types for the cells is at most k, and rules of the forms specified in op.

As well as this, we consider the families of languages of the form $ELP_n(tar)$ as the families of languages generated by standard rewriting P systems with at most n membranes where: the structure of the system is defined as a hierarchical arrangement of membranes represented as a tree, and communication of strings from one cell to another one is performed by using the target $here$, in_j, out. We refer to [10] for a formal definition of the basic model of rewriting P systems. Finally, we denote by MAT the family of languages generated by matrix grammars without appearance checking [6].

The first result we present here shows that unstructured population P systems without cell differentiation rules and cell division rules are no more than rewriting P systems.

Lemma 3.1. *$ELPP_{n,*}(\{a, b, c\}) \subseteq ELP_{n+1}(tar)$, for each $n \geq 1$.*

Proof. The proof is based on the observation that an unstructured population P system with $n \geq 1$ cells can be interpreted as being a rewriting P system with $n \geq 1$ elementary membranes embedded in a unique main membrane, which is the skin membrane of the P system and play the same role as the environment in the population P system. More precisely, given a population P system $\mathcal{P}$ with $n \geq 1$ cells $C_1, \ldots, C_n$ as specified in Definition 3.1, we construct a rewriting P system Π with $n + 1$ membranes labeled by $0, \ldots, n$ where:

- 0 is the label of the skin membrane;
- i, for each $1 \leq i \leq n$, is the label of an elementary membrane that is contained inside membrane 0;
- for each transformation rule $(\, a \to y\,)_i$ in R , with $1 \leq i \leq n$, there exists a corresponding rule $a \to (v, here)$ in R_i;
- for each output rule $(\, a\,)_i \to v\,(\,)_i$ in R, with $1 \leq i \leq n$, there exists a corresponding rule $a \to (v, out)$ in R_i;
- for each input rule $a(\,)_i \to (\, v\,)_i$ in R, with $1 \leq i \leq n$, there exists a rule $a \to (v, in_i)$ in R_0;

Notice that, without loss of generality, we are assuming $L = \{1, \ldots, n\}$ and each cell C_i to be of type i, for each $1 \leq i \leq n$. Moreover, in order to correctly simulate

the behaviour of $\mathcal{P}$, we add to R_0 a rule $a \to (a, out)$, for each symbol a in the system (i.e., at any moment a string produced inside the skin membrane can be sent out of the system), and we consider for the P system Π the set of terminal symbols $T' = \{\, a \mid a \in T, \nexists a(\,)_t \to (\,a\,)_i \in R \,\}$, for T the set of terminal symbols of $\mathcal{P}$ (i.e., we accept only strings in the environment that are terminal and that cannot enter anymore the cells in the system). This means the language $L(\Pi)$ containing all the the strings in T'^* sent out of the system during all the possible computations in Π is exactly the language $L(\mathcal{P})$. $\square$

Then, as a direct consequence of the equivalence $ELP_3(tar) = MAT$ established in the existing literature [7], [10] and the previous result, we obtain immediately the following result.

Corollary 3.1. $ELPP_{*,*}(\{a, b, c\}) \subseteq ELP_3(tar) = MAT$.

As well as this, we can prove the opposite inclusion: rewriting P systems can be simulated by unstructured population P systems without using cell differentiation and cell division.

Lemma 3.2. $ELP_n(tar) \subseteq ELPP_{n,n}(\{a, b, c\})$, *for each* $n \geq 1$.

Proof. Let Π be a rewriting P system with $n \geq 1$ membranes labeled in an one-to-one manner by $1, \ldots, n$ where 1 is the label of the skin membrane. We construct a population P system $\mathcal{P}$ with n cells where:

- $L = \{1, \ldots, n\}$;
- for each rule $a \to (v, here)$ in R_i, with $1 \leq i \leq n$, there exists a corresponding output rule $(\,a\,)_i \to v\$_{here_i}\,(\,)_i$ in R;
- for each rule $a \to (v, out)$ in R_i, with $1 < i \leq n$, there exists a corresponding output rule $(\,a\,)_i \to v\$_{out_i}\,(\,)_i$ in R;
- for each rule $a \to (v, out)$ in R_1, there exists a corresponding output rule $(\,a\,)_1 \to v\,(\,)_1$ in R;
- for each rule $a \to (v, in_j)$ in R_i, with $1 \leq i \leq n$, there exists a corresponding output rule $(\,a\,)_i \to v\$_{in_j}\,(\,)_i$ in R;
- there exists an input rule $\$_{here_i}\,(\,)_i \to (\,\lambda\,)_i$ in R, for each $1 \leq i \leq n$;
- there exists an input rule $\$_{in_i}\,(\,)_i \to (\,\lambda\,)_i$ in R, for each $1 \leq i \leq n$;
- there exists an input rule $\$_{out_j}\,(\,)_i \to (\,\lambda\,)_i$ in R, for each $1 \leq i \neq j \leq n$, with membrane i the membrane that contains membrane j.

The simulation of the P system Π by means of the population P system Π is done in the following way. We rewrite all the strings inside the cells in $\mathcal{P}$ in the same way as in Π by inserting in each of them a special symbol $\$_t$ carrying the target information t needed to move the strings in the right places. These strings are immediately moved out from the cells into the environment and, in the next step, they will be distributed

to the cells according to their respective target indications by using the input rules in R so that the process can be iterated. Moreover, the strings that are sent out of the skin membrane are moved into the environment without any special symbol $\$_t$, which means these strings will never be able to enter again the cells in the system. Therefore, the language $L(\mathcal{P})$ is exactly the language of terminal strings that are sent out of the system during all the possible computations in $\mathcal{P}$. □

Thus, by combining the previous result with Corollary 3.1, we obtain the following characterisation of the languages generated by unstructured population P systems without cell division rules and cell differentiation rules.

Corollary 3.2. $ELLP_{*,*}(\{a,b,c\}) = ELLP_{3,3}(\{a,b,c\}) = MAT.$

Finally, the case of population P systems with cell division rules and/or cell differentiation rules remains to be considered. In this respect, the sole operation of cell division is expected not to increase the power of population P systems, whereas population P systems with cell differentiation rules are expected to be computationally complete.

Conjecture 3.1. $ELLP_{*,*}(\{a,b,c,d\}) = RE.$

In fact, it should be easy to prove that population P systems with cell differentiation rules are able to simulate matrix grammars with appearance checking. This result, if proved, would be coherent with what was achieved in [1] where a similar universality result based on the operation of cell differentiation is provided for population P systems with symbol-objects.

4 Using Bond Making Rules

Direct communication among the cells (or membranes) in the system is one of the defining features of the membrane computing paradigm. Cells work in parallel on different sets (or multisets) of objects, which can then be moved from one cell to another one by means of some dedicated mechanisms typically expressed as a finite set of communication rules [10], [12], [8]. Moreover, population P systems [1] introduces the possibility of altering communication capabilities of the cells by modifying the set of links existing between the cells. This is done by considering a finite set of bond making rules which are used after each application of transformation/communication rules to the objects contained in the cells; this makes possible to modify the set of edges in the graph defining the structure of the system. Notice that a communication rule inside a cell can be used if and only if the cell is linked to some other cells by means of some edges in the underlying graph.

In grammar systems, these features of parallelism and communication among the components were firstly considered by introducing the notion of parallel communicating grammar systems (PC grammar systems, for short): a model for networks of Chomsky grammars communicating strings by emerging requests [3]. More precisely, in each step, each grammar in the system rewrites its string and communication is done by requests through so-called query symbols, each one of them referring to a

specific grammar in the system. When a query symbol appears in the string of a grammar, the rewriting process stops and one or more communication steps are performed by replacing all occurrences of the query symbols with the current string of the queried grammars providing that this string does not contain any query symbol. When no more query symbols are present in the system the rewriting process can start again.

From a P system point of view, this communication by request can be considered as a mechanism to open communication channels between the cells in the system. Specifically, here we want to use query symbols for defining bond making rules that make possible to link two cells provided that these cells contain a pair of corresponding query symbols. Then, once a link has been established, communication between the two cells can take place by rewriting their respective query symbols by means of some particular communication rules associated with the cells.

Definition 4.1. A population P system with bond making rules is a construct

$$\mathcal{P} = (V, T, Q, \alpha, C_1, C_2, \ldots, C_n)$$

where:

1. V is a finite alphabet;
2. $T \subseteq V$ is a finite set of terminal symbols;
3. Q is a finite of set of query symbols;
4. α is a finite set of bond making rules of the form $(i, q_1; q_2, j)$, with $1 \leq i \neq j \leq n$ and $q_1, q_2 \in Q$;
5. $C_i = (M_i, R_i, S_i)$, for each $1 \leq i \leq n$, with:
   (a) $M_i \subseteq V^*$ a finite language defining the initial content of cell i;
   (b) R_i a finite set of transformation rules of the forms $a \to v$, $a \to v\,q$, $a \to (v, out)$, for $a \in V$, $v \in V^*$, and $q \in Q$;
   (c) S_i is a finite set of communication rules of the forms $q \to v$, $q \to v\,q'$, for $q, q' \in Q$, and $v \in V^*$.

Here, with respect to Definition 3.1, we are considering population P systems without cell differentiation and cell division where the number and the type of cells in the system cannot vary during a computation. As well as this, we do not need a notion of environment as cells are allowed to communicate directly by using their respective sets of communication rules in combination with the set of bond making rules. In this respect, the structure of a population P system with bond making rules $\mathcal{P}$ is given by the the set of communication links existing among the cells in the system that are created time by time by applying the bond making rules in α.

Each cell C_i, with $1 \leq i \leq n$, gets assigned a finite language M_i that defines its initial content. Each string contained in cell i is rewritten by using the transformation rules in R_i. A rule $a \to v \in R_i$ specifies that a string containing a symbol a can be rewritten inside cell i by replacing the symbol a with the string v. In a

similar way, a rule $a \to (v, out) \in R_i$ allows cell i to rewrite one of its internal strings but, after that, the string has to exit the cell without any chance to enter again any cell of the system. Finally, a rule $a \to v\,q \in R_i$ makes possible to rewrite a string inside cell i and introduce in this string the query symbol q. After the application of one of these rules, the rewriting of the string containing the query symbol q inside cell i stops and it will restart only when the symbol q has been satisfied and removed from the string. However, this does not stop the rewriting of the other strings contained in cell i. In order to satisfy a query symbol, the system operates as follows. Consider a bond making rule $(i.q_1; q_2, j)$, with $q_1, q_2 \in Q$, $1 \leq i \neq j \leq n$, and suppose that, after one application of the respective transformation rules in R_i and in R_j, a string $x\,q_1\,y$ is produced inside cell i and a string $u\,q_2\,z$ is produced inside cell j, with $x, y, u, z \in V^*$. This means a bond between cell i and cell j can be created by connecting the string $x\,q_1\,y$ with the string $u\,q_2\,z$, and the query symbol q_1 can be satisfied by using the communication rules in S_j whereas the symbol q_2 can be satisfied by using the rules in S_i. Notice that, for each $1 \leq k \leq n$, the set S_k contains both rules of the form $q \to v$ that remove the query symbol from the string, and rules of the form $q \to v\,q'$ that introduce a new query symbol, which makes the string to wa!it for another communication to take place.

As usual, we assume a non-deterministic maximal parallel strategy for the application of the rules:

- in each step, in each cell i, with $1 \leq i \leq n$, each string that can be rewritten by some rules in R_i must be rewritten by using one rule in R_i non-deterministically chosen (i.e., each string inside cell i is rewritten in a sequential manner); at the same time, each query symbol in each cell j linked to cell i that can be satisfied must be satisfied by using one communication rule in S_i non-deterministically chosen;

- after each application of the transformation/communication rules, for each bond making rule $(i.q_1; q_2, j)$, with $q_1, q_2 \in Q$, $1 \leq i \neq j \leq n$, we establish a link between cell i and cell j for each pair of strings $x\,q_1\,y$, $u\,q_2\,z$, with the former one contained in cell i and the latter one contained in cell j, for $x, y, u, z \in V^*$; the restriction here is that the same string containing a query symbol can be used by only one bond making rule non-deterministically chosen.

Therefore, a step of computation in a population P system with bond making rules is done in two separate stages: a stage where transformation/communication rules are applied to the strings contained inside the cells and a stage where bond making rules are used in order to create links between the cells in the system. Notice that the query symbols introduced by the application of the transformation rules can be satisfied only in the next step of computation after having used the bond making rules. Furthermore, these bond making rules make possible to create multi-links between the cells that are represented by sets of pairs of corresponding strings.

The language generated by a population P system with bond making rules $\mathcal{P}$ is the language $L(\mathcal{P})$ containing all the strings sent out of the cells during all the possible computations in $\mathcal{P}$. The family of languages generated by population P systems with bond making rules and with at most $n \geq 1$ cells is denoted by $ELPP_n(query)$.

Next, we prove that these population P systems are quite powerful from a computational point of view: systems with only two membranes can generate the whole family of languages in MAT.

Lemma 4.1. $MAT \subseteq ELPP_2(query)$.

Proof. Consider, without loss of generality, a matrix grammar without appearance checking G in binary normal form. This is because it is known from [6] that, for each matrix grammar generating a language L, there always exists an equivalent matrix grammar in binary normal formal generating the same language L. Moreover, we assume all the matrices in G but the initial matrix $(S \to XA)$ to be labeled in an one-to-one manner by values in $\{1, 2, \ldots, m\}$. We construct a population P system $\mathcal{P}$ with 2 cells where:

- $C_1 = (M_1, R_1, S_1)$ with
  - $M_1 = \{X\}$, for $(S \to XA)$ the initial matrix of G;
  - $R_1 = \{\, X \to X_i \mid X \in N_1, 1 \leq i \leq n \,\}$;
  - $S_1 = \{\, A_i \to v \mid i : (X \to Y, A \to v)$ is a matrix in $G, 1 \leq i \leq n \,\}$ $\cup \{\, A_i \to v\, f \mid i : (X \to \lambda, A \to v)$ is a matrix in $G, 1 \leq i \leq n \,\}$;
- $C_2 = (M_2, R_2, S_2)$ with
  - $M_2 = \{A\}$, for $(S \to XA)$ the initial matrix of G;
  - $R_2 = \{\, A \to A_i \mid A \in N_2, 1 \leq i \leq n \,\} \cup \{\, f \to (f, out) \,\}$;
  - $S_2 = \{\, X_i \to Y \mid i : (X \to Y, A \to v)$ is a matrix in $G, 1 \leq i \leq n \,\}$ $\cup \{\, X_i \to \lambda \mid i : (X \to \lambda, A \to v)$ is a matrix in $G, 1 \leq i \leq n \,\}$;

for $Q = \{\, X_i, A_i \mid X \in N_1, A \in N_2, 1 \leq i \leq n \,\}$. Now, by having this construction, it is easy to see that the population P system $\mathcal{P}$ correctly simulates the matrix grammar G and therefore we have $L(\mathcal{P}) = L(G)$. □

At the moment, we are not able to provide an upper bound for the generative capacity of population P systems with bond making rules and answer the question whether they are more than usual rewriting P systems or not.

5 Final Remarks

Membrane computing and grammar systems are two active areas of theoretical computer science, with different starting points, but with several similarities (both areas deal with distributed computing devices, where such notions as parallelism, cooperation, decentralisation are crucial). Nevertheless, as we have seen, important differences between the two models emerge from a deeper investigation especially in terms of computational power. Population P systems even of a very basic form are able to characterise the whole family of languages generated by matrix grammars without appearance checking (see Corollary 3.2). A similar result was already established for the basic model of rewriting P systems as reported in Corollary 3.1. The power

of (population) P systems is in fact determined by the underlying communication structure that allows the cells to exchange strings in a controlled way. In this sense, P systems with string-objects appears to be more similar to the already existing notions of networks of evolutionary processors [5], [2].

On the other hand, a general claim in the area of P systems with string-objects is that rewriting is not enough to achieve the computational completeness but it is necessary to introduce in rewriting P systems some extra features such as priority, replicated rewriting or conditional communication [10]. In the case of unstructured population P systems, we expect the universality to be obtained by including the operation of cell differentiation, which makes possible to change the types of the cells in the system (see Conjecture 3.1). Furthermore, the claim "rewriting is not enough" might represent a motivation for introducing grammar system features in (population) P systems. In this respect, Section 4 introduces a variant of population P systems where the notion of query symbols considered for PC grammar systems is used to define bond making rules and the related communication model. Notice that, according to Definition 4.1, communication in a population P system with bond making rules consists in just rewriting a query symbol in a cell by means of some rules associated with another cell without an effective movement of the strings from a cell to another one. In other words, a string containing a query symbol is temporarily given access to the rules in a cell different from the one where the string is placed. Nevertheless, we could easily define a variant of this model where strings are moved from a cell to another one by replacing a query symbol in a cell with all the strings contained in the queried cell and producing all the resulting strings inside the querying cell. As well as this, we might consider multisets of strings instead of sets of strings as, for instance, proposed in [11] where PC grammar systems with multisets of strings are investigated in relationship with P systems.

Finally, further investigations of population P systems with string-objects might be directed to clarify the role of the environment in achieving successful and meaningful computation; this might be done by pointing out analogies with the existing models of eco-grammar systems [4].

Acknowledgements

This research was supported by the Molecular Computing Network (MolCoNet), European Union Contract IST-2001-32008 and by the Engineering and Physical Science Research Council (EPSRC) of United Kingdom, Grant GR/R84221/01.

References

[1] Bernardini, F., Gheorghe, M., (2004). Population P systems. *Journal of Universal Computer Science*, **10**, 509-539.

[2] Castellanos, J., Martin-Vide, C., Mitrana, V., Sempere, J., M., (2003). Networks of Evolutionary Processors. *Acta Informatica*, **39**, 517-529.

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

[4] Csuhaj-Varjú, E., Kelemen, J., Kelemenova, A., Păun, Gh., (1997). Eco-Grammar Systems: A Grammatical Framework for Studying Life-Like Interactions. *Artificial Life*, **3**, 1-28.

[5] Csuhaj-Varjú, E., Salomaa, A., (1997). Networks of Parallel Languages Processors. In *New Trends in Formal Languages* (Pň, Gh., Salomaa, A., eds.), Lecture Notes in Computer Science, **1218**, Springer, Berlin, Heidelberg, New York, 299-318.

[6] Dassow, J., Păun, Gh., (1989). *Regulated Rewriting in Formal Language Theory.* EATCS Monograph in Theoretical Computer Science, Springer-Verlag, Berlin, Heidelberg, New York.

[7] Madhu, M., (2003). *Studies of P Systems as a Model of Cellular Computing*, PhD Thesis, Indian Institute of Technology, Madras, India.

[8] Martin-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A., eds., (2004). *Membrane computing. International workshop, WMC 2003, Tarragona, Spain, July 2003. Revised papers.* Lecture Notes in Computer Science, **2933**, Springer, Berlin, Heidelberg, New York.

[9] Păun, Gh., (2000). Computing with Membranes. *Journal of Computer and System Sciences*, **61**, 108–143.

[10] Păun, Gh. (2002). *Membrane Computing. An Introduction.* Natural Computing Series, Springer, Berlin, Heidelberg, New York.

[11] Păun, Gh., (2004). Grammar Systems vs. Membrane Computing: A Preliminary Approach. In *Pre-Proceedings of Grammar Systems Week 2004, Budapest, Hungary, July 5-9, 2004* (Csuhaj-Varjú, E., Vaszil, G., eds.), MTA SZTAKI, Budapest, 225-244.

[12] Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C., eds., (2003). *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19-23, 2002. Revised papers.* Lecture Notes in Computer Science, **2597**, Springer, Berlin, Heidelberg, New York.

[13] Rozenberg, G., Salomaa, A., eds., (1997). *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, Heidelberg, New York.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 78 - 94, 2004.

CD Grammar Systems as Models of Distributed Problem Solving, Revisited

Henning Bordihn

Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, D-14482 Potsdam, Germany
henning@cs.uni-potsdam.de

Markus Holzer

Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
holzer@informatik.tu-muenchen.de

Abstract

Based on a derivation mode f for cooperating distributed (CD) grammar systems, we introduce a new form of cooperation protocol, the so called "cut-f-mode" of derivation. Intuitively, a cut-f-mode derivation, partitions (cuts) a sentential form into several subwords, where some of these subwords are distributed to the components, which derive words according to the original f-mode of derivation, and finally combines all these words together again. It is argued that these derivation modes are much closer to the original AI motivation of CD grammar systems. We investigate the cut-mode versions of the classical derivation modes $*$, $\leq k$, $=k$, and $\geq k$, of the competence based modes t and sf, as well as the cut-mode versions of the combined t-modes. It turns out that in most cases, the cut-f-mode turns out to be as most as powerful than the corresponding non-cut-mode, that is, the f-mode itself. Nevertheless, there are also some cases, where the power is even reduced to that of context-free grammars.

1 Introduction

The theory of cooperating distributed grammar systems—for an overview we refer to [7]—has become a vivid field in formal language theory since its origin in [4], with forerunner papers [10] and [1]. Cooperating distributed grammar systems have been introduced for describing, in terms of formal grammars and languages, communities of cooperating autonomous problem solving agents which use the blackboard model of problem solving. Furthermore, grammar systems are motivated in connection with the syntax of programming languages, since they can be seen as generalization of two-level substitution grammars to a multi-level concept [10]. Finally, in recent papers they have been considered as sequential counterparts of tabled Lindenmayer systems [2].

A cooperating distributed (CD, for short) grammar system consists of a finite set of (context-free) grammars, called components, performing derivation steps on a common sentential form in turns, according to some cooperation protocol. In terms of the blackboard model of problem solving, the components correspond to the independent problem solving agents, representing autonomous knowledge sources, the sentential form to the current state of the problem solving (the blackboard), where the knowledge sources can make changes, and the generated language represents the set of problem solutions (cf. [5]). Simple cooperation protocols are the so-called $*$-mode, $\leq k$-mode, $=k$-mode, or $\geq k$-mode, where a component, once started, has to perform an arbitrary number, at most k, exactly k, or at least k derivation steps, respectively. Moreover, there are two cooperation protocols which are based on the feature of competence of the problem solving agents on the current state of the problem solving: (1) In the t-mode, a component can start and has to remain deriving unless and until there is no nonterminal left in the sentential form to which one of its productions is applicable (that is, the component is not able to contribute to the problem solving any more), and (2) in the sf-mode, a component is allowed to become and has to remain active unless and until there is some nonterminal present in the sentential form which cannot be rewritten by this component (that is, the component does not possess the full competence on the current state of the problem solving).

We try to model the effect when agents contribute to the solution by solving subtasks of the whole problem. From the AI motivation this approach seems to be more adequate: In contrast to different rule-based architectures of problem solving systems, the blackboard model of problem solving emphasizes the highly opportunistic way in which the knowledge sources are applied during the problem solving process, cf. [11]. That is, there is no rigid prescription for the cooperation strategy of the knowledge sources. In terms of CD grammar systems, solving sub-tasks corresponds to working on a substring of the current sentential form. This led the authors to the concept of CD grammar systems working in the *cut-f-mode* of derivation, where f is one of the aforementioned cooperation protocols. In these *cut-f-modes*, in any derivation step the sentential form is partitioned (cut) into several substrings which can be associated to different components. In order to be as little rigid as possible, this association is done via a partial mapping, such that both some substrings of the sentential form and some components may be disregarded. Then, each component to which a substring has been associated works in (one and the same) derivation mode, more precisely, in the f-mode of derivation if the CD grammar system as a whole is driven in the *cut-f-mode*.

The overall picture that emerges from our investigates on this newly defined derivation mode is, that for some modes, the generative capacity is not affected at all, but there are also some cases, where the power is reduced to that of context-free grammars, if parts of the sentential forms can be distributed to several components in one and the same derivation step. In particular, the former is true for derivations like, e.g., $*$-, $=1$-, ≥ 1-, or ≤ 1-mode, while the latter is true for instance, for the t-mode of derivation, which describes the family of all ET0L languages in the non-cut variant. Moreover, we also find situations, where the cut-mode derivation does not become context-free. For instance, the sf-mode of derivation characterizes the

family of programmed context-free languages, while its cut-mode version describes an intermediate class between the families of recurrent programmed context-free and programmed context-free languages. The former language family equals the biologically motivated family of languages generated by ET0L systems with random context [15]. This is yet another result, where recurrent programmed languages appear in relation with CD grammar systems and unconventional derivation modes—see, e.g., [3].

The paper is organized as follows: The next section contains preliminaries, and we provide the basic definition of cut-f-mode derivations. Then in Section 3 we investigate the generative power of CD grammar systems working in these newly defined derivations modes and finally, we summarize our results and highlight the remaining open questions in Section 4.

2 Definitions

We assume the reader to be familiar with the basic notions of formal languages, as contained in [6]. In general, we have the following conventions: $\subseteq$ denotes inclusion, while $\subset$ denotes strict inclusion. The set of positive integers is denoted by $\mathbb{N}$ and the cardinality of a set M by $|M|$. Concerning our set notation, we abbreviately write $\{ X(k), Y(k) \mid P(k) \}$ instead of $\{ X(k) \mid P(k) \} \cup \{ Y(k) \mid P(k) \}$, where $X(k)$ and $Y(k)$ are objects with parameter k and $P(k)$ is some predicate on k. This notation is also extended to more than two kinds of objects having the same parameter. The empty word is denoted by λ. For $x \in V^*$, where V is some alphabet, and for $W \subseteq V$, let $|x|_W$ denote the number of occurrences of letters from W in x. If W is a singleton set $\{a\}$, we simply write $|x|_a$ instead of $|x|_{\{a\}}$. We consider two languages L_1 and L_2 to be equal if and only if $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, and we simply write $L_1 = L_2$ in this case.

The families of languages generated by regular, context-free, context-sensitive, general type-0 Chomsky grammars, and ET0L systems are denoted by $\mathcal{L}(\mathrm{REG})$, $\mathcal{L}(\mathrm{CF})$, $\mathcal{L}(\mathrm{CS})$, $\mathcal{L}(\mathrm{RE})$, and $\mathcal{L}(\mathrm{ET0L})$, respectively. We attach $-\lambda$ in our notations if erasing rules are not permitted. Details about these families can be found in [6]. The class of finite languages is denoted by $\mathcal{L}(\mathrm{FIN})$.

A *programmed grammar* (see, for instance, [6, 13]) is a septuple

$$G = (N, T, P, S, \Lambda, \sigma, \phi),$$

where N, T, and $S \in N$ are the set of nonterminals, the set of terminals, and the start symbol, respectively. Here P is the finite set of productions of the form $\alpha \to \beta$, $\alpha, \beta \in (N \cup T)^*$, $|\alpha|_N > 0$, and Λ is a finite set of labels (for the productions in P), such that Λ can be also interpreted as a function which outputs a production when being given a label; σ and ϕ are functions from Λ into the set of subsets of Λ. Usually, the productions are written in the form

$$(r:\ \alpha \to \beta,\ \sigma(r),\ \phi(r)),$$

where r is the label of $\alpha \to \beta$. For (x, r_1) and (y, r_2) in $(N \cup T)^* \times \Lambda$ and $\Lambda(r_1) = (\alpha \to \beta)$, we write $(x, r_1) \Rightarrow (y, r_2)$ if and only if either $x = x_1 \alpha x_2$, $y = x_1 \beta x_2$ and

$r_2 \in \sigma(r_1)$, or $x = y$ and rule $\alpha \to \beta$ is not applicable to x, and $r_2 \in \phi(r_1)$. In the latter case, the derivation step is done in *appearance checking mode*. The set $\sigma(r_1)$ is called success field and the set $\phi(r_1)$ failure field of r_1. As usual, the reflexive transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. The language generated by G is defined as

$$L(G) = \{\, w \in T^* \mid (S, r_1) \overset{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \Lambda \,\}.$$

The family of languages generated by programmed grammars containing only context-free core rules is denoted by $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac})$. We replace CF by $\mathrm{CF} - \lambda$ in that notation if erasing rules are forbidden. When no appearance checking features are involved, i.e., $\phi(r) = \emptyset$ for each label $r \in \Lambda$, we are led to the families $\mathcal{L}(\mathrm{P}, \mathrm{CF})$ and $\mathcal{L}(\mathrm{P}, \mathrm{CF} - \lambda)$. A special variant of programmed grammars are recurrent programmed grammars introduced in [15]. A programmed context-free grammar G is a *recurrent programmed context-free grammar* if for every $p \in \Lambda$ of G, if $\phi(p) = \emptyset$, then $p \in \sigma(p)$, and if $\phi(p) \neq \emptyset$, then $p \in \sigma(p) = \phi(p)$. The corresponding language families are denoted by $\mathcal{L}(\mathrm{RP}, \mathrm{CF}, \mathrm{ac})$ and $\mathcal{L}(\mathrm{RP}, \mathrm{CF} - \lambda, \mathrm{ac})$. When no appearance checking features are involved, i.e., $\phi(r) = \emptyset$ for each label $r \in \Lambda$, we omit ac in that notation, again.

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

Moreover, we need some more notation on grammar, namely we have to define the *finite index* restriction. Loosely speaking, the index of a grammar is the maximal number of nonterminals simultaneously appearing in a sentential form during a terminating derivation, considering the most economical derivation for each string. The finite index property is defined as follows: Let G be an arbitrary grammar type (e.g., context-free grammars, programmed grammars with or without appearance checking, etc.), and let N, T, and $S \in N$ be its nonterminal alphabet, terminal alphabet, and axiom, respectively. For a derivation

$$D : S = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = w$$

for w in G, with $w \in T^*$, we set

$$ind(D, G) = \max\{\, |w_i|_N \mid 1 \leq i \leq n \,\},$$

and, for $w \in T^*$, we define

$$ind(w, G) = \min\{\, ind(D, G) \mid D \text{ is a derivation for } w \text{ in } G \,\}.$$

The *index of grammar* G is defined as

$$ind(G) = \sup\{ind(w, G) \mid w \in L(G) \,\}.$$

For a language L in the family $\mathcal{L}(X)$ of languages generated by grammars of type X we define

$$ind_X(L) = \inf\{\, ind(G) \mid L(G) = L \text{ and } G \text{ is of type } X \,\}.$$

For a family $\mathcal{L}(X)$, we set

$$\mathcal{L}_n(X) = \{\, L \mid L \in L(X) \text{ and } ind_X(L) \leq n \,\}$$

for $n \in \mathbb{N}$, and $\mathcal{L}_{fin}(X) = \cup_{n\geq 1}\mathcal{L}_n(X)$.

A *cooperating distributed (CD) grammar system* of degree n, with $n \geq 1$, is an $(n+3)$-tuple $G = (N, T, P_1, P_2, \ldots, P_n, S)$, where N, T are disjoint alphabets of nonterminals and terminals, respectively, $S \in N$, and $P_1, P_2, \ldots, P_n$ are context-free rule sets called components. For $1 \leq i \leq n$, let

$$\mathrm{dom}(P_i) = \{\, A \in N \mid \text{there is a word } v \text{ such that } A \to v \in P_i \,\}$$

denote the set of all nonterminals which can be rewritten by the component P_i. For $x, y \in (N \cup T)^*$ and $1 \leq i \leq n$, we write $x \Rightarrow_i y$ if and only if $x = x_1Ax_2$ and $y = x_1zx_2$ for some $A \to z \in P_i$. Hence, subscript i refers to the component to be used. By $\Rightarrow_i^{\leq k}$, $\Rightarrow_i^{=k}$, $\Rightarrow_i^{\geq k}$, $\Rightarrow_i^*$, for $k \geq 1$, we denote a derivation consisting of at most k steps, exactly k steps, at least k steps, an arbitrary number of steps, respectively, executed by component P_i. Furthermore, we write $x \Rightarrow_i^t y$ if and only if $x \Rightarrow_i^* y$ and there is no z such that $y \Rightarrow_i z$. Moreover, $x \Rightarrow_i^{sf} y$ if and only if $x \Rightarrow_i^* x'$, $x' \Rightarrow_i y$, and P_i is sf-competent on x' but is *not* sf-competent on y, where a component P_i is said to be *sf-competent* on a word x if and only if (1) $x = u_0A_1u_1A_2u_2 \ldots u_{m-1}A_mu_m$ with $m \geq 0$, $u_j \in T^*$, for $0 \leq j \leq m$, and $A_j \in N$, for $1 \leq j \leq m$, and (2) for each j, for $1 \leq j \leq m$, there is a production $A_j \to w_j$ in P_i. Note that the definition of the derivation relation implies that component P_i is sf-competent on x and on all intermediate sentential forms in the derivation $x \Rightarrow_i^* x'$, too.

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—see, e.g., [8, 9]: There exists a derivation which satisfies both properties in common, 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 $y \Rightarrow_i z$.

Applying our idea on distributed problem solving to one of the aforementioned modes

$$f \in \{*, t, sf\} \cup \{\leq k, =k, \geq k \mid k \in \mathbb{N}\} \cup \{\, (t \wedge \leq k), (t \wedge = k), (t \wedge \geq k) \mid k \in \mathbb{N} \,\}$$

leads us to the *cut-f-mode*, f_c-mode for short, which is defined as follows: $x \Rightarrow^{f_c} y$ if and only if

1. $x = x_0x_1 \ldots x_m$ with $m \geq 0$, $x_i \in (N \cup T)^*$, for $0 \leq i \leq m$,
2. there is a partial injective mapping $\rho : \{0, 1, \ldots m\} \hookrightarrow \{1, 2, \ldots, n\}$ such that $y_i = x_i$, if $i \notin \mathrm{dom}(\rho)$, and $y_i = z_i$, if $x_i \Rightarrow_{\rho(i)}^f z_i$, and
3. $y = y_0y_1 \ldots y_m$.

Here $\mathrm{dom}(\rho) = \{\, i \mid \rho(i) \text{ is defined} \,\}$ denotes the set of indices in the decomposition $x = x_0x_1 \ldots x_m$ to which some component is associated by ρ. Let

$$D = \{*, t, sf\} \cup \{\leq k, =k, \geq k \mid k \in \mathbb{N}\} \cup \{\, (t \wedge \leq k), (t \wedge = k), (t \wedge \geq k) \mid k \in \mathbb{N} \,\}$$

and $D_c = \{ f_c \mid f \in D \}$. The language generated by the CD grammar system

$$G = (N, T, P_1, P_2, \ldots, P_n, S)$$

in the *non-cut* f-mode with $f \in D$ is defined as

$$L_f(G) = \{ w \in T^* \mid S \Rightarrow^f_{i_1} w_1 \Rightarrow^f_{i_2} \cdots \Rightarrow^f_{i_{m-1}} w_{m-1} \Rightarrow^f_{i_m} w_m = w \text{ with } m \geq 1,\ 1 \leq i_j \leq n, \text{ for } 1 \leq j \leq m \}$$

whereas the the language generated in the cut-f-mode with $f_c \in D_c$ is defined as

$$L_{f_c}(G) = \{ w \in T^* \mid S \Rightarrow^{f_c} w_1 \Rightarrow^{f_c} \cdots \Rightarrow^{f_c} w_{m-1} \Rightarrow^{f_c} w_m = w \text{ with } m \geq 1 \}.$$

If $f \in D \cup D_c$, then the family of languages generated by [λ-free] context-free CD grammar systems working in f-mode, is denoted by $\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f)$.

In order to clarify our definitions, we give two examples.

Example 1. *Let G be the CD grammar system $G = (N, T, P_1, P_2, P_3, S)$ with non-terminals $N = \{S, S', A, A', B, B'\}$, terminals $T = \{a, b, c\}$, and the production sets*

$$\begin{aligned} P_1 &= \{S \to S', S' \to AB\} \\ P_2 &= \{A \to aA'b, B \to B'c\} \\ P_3 &= \{A' \to A, B' \to B\} \\ P_4 &= \{A \to ab, B \to c\}. \end{aligned}$$

It is easy to see that running G in the $=2$-mode results in the non-context-free language $L_1 = \{ a^n b^n c^n \mid n \geq 1 \}$, since the only way to start the derivation is to use production set P_1 leading to the sentential form AB within two steps. Then, for all natural numbers $n \geq 0$, we find

$$a^n A b^n B c^n \Rightarrow^{=2}_2 a^{n+1} A' b^{n+1} B' c^{n+1} \Rightarrow^{=2}_3 a^{n+1} A b^{n+1} B c^{n+1}$$

or the terminating derivation

$$a^n A b^n B c^n \Rightarrow^{=2}_4 a^{n+1} b^{n+1} c^{n+1}.$$

This shows the stated claim. Observe, that except for the axiom and the terminal word, all intermediate sentential forms contain exactly two *nonterminals. When considering the $(=2)_c$-mode a similar reasoning applies, since no production set is successfully applicable to a sentential form containing* one *nonterminal only, except from the application of P_1 to the axiom S. Thus, a production set P_i with $2 \leq i \leq 4$ will be successful only, if two appropriate nonterminals in the sentential form are present. Therefore, the only possible derivations in $(=2)_c$-mode are the derivations shown above.*

When comparing the t-mode and its cut-version we find that in the first mode the CD grammar system G generates the language $\{ a^n b^n c^n \mid n \geq 1 \}$ while in t_c-mode only the context-free language $\{ a^n b^n c^m \mid n, m \geq 1 \}$ will be obtained. The latter fact is obvious, since due to the cutting of the sentential form when running G in the t_c-mode the derivation of the two nonterminals will be decoupled. Therefore, the number of a's and b's are independently from the number of c's in the sentential form. Thus, we obtain the context-free language mentioned above.

Let us come to our second example.

Example 2. *The language* $L_1 = \{\, a^n b^n c^n \mid n \geq 1 \,\}$ *is also generated by the CD grammar system* $G = (\{S, A, A', B, B'\}, \{a, b, c\}, P_1, P_2, P_3, P_4, S)$ *with the production sets*

$$\begin{aligned} P_1 &= \{S \to AB\} \\ P_2 &= \{A \to aA'b, A' \to A', B \to B'c\} \\ P_3 &= \{A' \to A, B' \to B, B \to B\} \\ P_4 &= \{A \to ab, B \to c\}, \end{aligned}$$

if it is driven in either sf- or sf_c*-mode. For the sf-mode, this is seen as follows. Every derivation starts with* P_1 *yielding* AB*. Starting with a sentential form* $a^n Ab^n Bc^n$*,* $n \geq 0$*, the derivation can terminate to* $a^{n+1}b^{n+1}c^{n+1}$ *with the help of* P_4 *or* P_2 *can be applied. Then, we have to distinguish two cases: (1)* $a^n Ab^n Bc^n \Rightarrow_2^{sf} a^{n+1}A'b^{n+1}B'c^{n+1}$ *by replacing* A *first and (2)* $a^n Ab^n Bc^n \Rightarrow_2^{sf} a^n Ab^n B'c^{n+1}$ *by applying* $B \to B'c$*, first and only. In the second case, the derivation is blocked, since no of the components is competent on the obtained sentential form. In the first case,* P_3 *is the only applicable component. If* A' *is replaced by* A *first, then the derivation is blocking again, since no component is competent both on* A *and on* B'*. Hence, first* B' *and then* A' *must be rewritten, yielding* $a^{n+1}Ab^{n+1}Bc^{n+1}$*. In conclusion,* $L_{sf}(G) = L_1$*.*

The fact $L_{sf_c}(G) = L_1$ *can similarly be seen if one takes into consideration the following observations:*

1. P_2 *cannot work on a string containing an* A *but no* B *since it would not terminate.*

2. *If* P_2 *deals with a string containing a* B *and no* A*, then the* B' *contained in the resulting string cannot be replaced, since* P_3 *can only terminate if it starts off with a string containing an* A'*, but this* A' *cannot be produced by* P_2 *after* B *has been replaced before (see item 1.).*

3. *Analogously,* P_3 *can successfully work neither on a string containing a* B' *but no* A' *nor on a string containing an* A' *but no* B'*.*

3 The Power of Cut-Derivations in CD Grammar Systems

In this section we focus on the power of the newly defined CD grammar systems variants with the well known "classical" language families as introduced in the preceding section. First, we recall some known facts about the generative capacity of CD grammar systems working in the non-cut modes, see, for example, [7]. For $f \in \{*\} \cup \{=1, \geq 1\} \cup \{\leq k \mid k \geq 2\}$, CD grammar systems with context-free components characterize exactly the context-free languages, that is, $\mathcal{L}(\mathrm{CF}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f)$, while those CD grammar systems working in f-mode with $f \in \{\, =k, \geq k \mid k \geq 2 \,\}$

induce a proper superset of the family of context-free languages which is contained in the family of all context-free programmed languages without appearance checking. In other words, $\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f) \subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda])$, if $f \in \{=k, \geq k \mid k \geq 2\}$. It is unknown whether the second inclusion is strict. The next theorem shows that for the aforementioned derivation modes, there is no difference to its cut version.

Theorem 3. *Let* $f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \in \mathbb{N}\}$. *Then*

$$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

Proof. Let $G = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system. We show that $L_f(G) = L_{f_c}(G)$. For the inclusion from left to right it is sufficient to argue that any f-mode derivation $x \Rightarrow_f y$ can be simulated by $x \Rightarrow^{f_c} y$ choosing $x = x_0$ and $\rho : \{0\} \hookrightarrow \{1, 2, \ldots, n\}$ to be the constant function $\rho(0) = i$. Conversely we argue that a f_c-derivation step can be sequentialized. This is seen as follows: Let $x \Rightarrow^{f_c} y$ with (1) $x = x_0 x_1 \ldots x_m$, (2) the injective partial mapping $\rho : \{0, 1, \ldots, m\} \hookrightarrow \{1, 2, \ldots, n\}$, and (3) $y = y_0 y_1 \ldots y_m$ fulfilling $y_i = x_i$, if $i \notin \mathrm{dom}(\rho)$, and $y_i = z_i$, if $x_i \Rightarrow^f_{\rho(i)} z_i$. To simplify presentation, let $\mathrm{dom}(\rho) = \{i_1, i_2, \ldots, i_r\}$ with $0 \leq i_1 \leq i_2 \leq \ldots \leq i_r \leq m$. Observe, that $y_i = x_i$ if $i \in \{0, 1, \ldots, m\} \setminus \mathrm{dom}(\rho)$. Then we find an f-mode derivation

$$\begin{aligned}
x &= y_0 y_1 \ldots y_{i_1-1} x_{i_1} y_{i_1+1} \ldots y_{i_2-1} x_{i_2} y_{i_2+1} \ldots y_m \\
&\Rightarrow^f_{\rho(i_1)} \quad y_0 y_1 \ldots y_{i_1-1} y_{i_1} y_{i_1+1} \ldots y_{i_2-1} x_{i_2} y_{i_2+1} \ldots y_m \\
&\Rightarrow^f_{\rho(i_2)} \quad y_0 y_1 \ldots y_{i_2-1} y_{i_2} y_{i_2+1} \ldots y_m \\
&\Rightarrow^f_{\rho(i_3)} \quad \ldots \\
&\Rightarrow^f_{\rho(i_r)} \quad y_0 y_1 \ldots y_m = y,
\end{aligned}$$

which simulates the original f_c-mode derivation. This proves $L_f(G) = L_{f_c}(G)$. □

An immediate consequence of the above given theorem is the following corollary.

Corollary 4. *1. If* $f \in \{*\} \cup \{=1, \geq 1\} \cup \{\leq k \mid k \geq 2\}$, *then*

$$\mathcal{L}(\mathrm{CF}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

2. If $f \in \{=k, \geq k \mid k \geq 2\}$, *then*

$$\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c) \subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda]).$$

Next we turn our attention to the t-mode. For CD grammar systems working in the t-mode it was shown (see [5]) that

$$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], t) = \mathcal{L}(\mathrm{ET0L})$$

and thus they build a strict superset of the family of context-free languages. In contrast to this ET0L characterization, we show that CD grammar systems running in the cut-t-mode, characterize only the family of context-free languages.

Theorem 5. $\mathcal{L}(\mathrm{CF}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], t_c)$.

Proof. The inclusion from left to right holds since any context-free grammar can be interpreted as a CD grammar system with one component, generating the same language in t_c-mode. For the converse inclusion, let a CD grammar system $G = (N, T, P_1, P_2, \ldots, P_n, S)$ be given. For $1 \leq i \leq n$, set $N^{(i)} = \{ A^{(i)} \mid A \in \mathrm{dom}(P_i) \}$ and let $g_i : N \cup T \to N \cup T \cup N^{(i)}$ be the morphism defined by $g_i(X) = X^{(i)}$ if $X \in \mathrm{dom}(P_i)$ and $g_i(X) = X$ otherwise. Consider the CD grammar system

$$G' = (N', T, P'_1, P'_2, \ldots P'_n, S),$$

where $N' = N \cup \bigcup_{i=1}^{n} N^{(i)}$ and, for $1 \leq i \leq n$,

$$P'_i = \{ A \to g_i(v),\ A^{(i)} \to g_i(v) \mid A \to v \in P_i \}.$$

Observe that the axiom of G' is contained in N and that, for any string $x \in (N')^*$ and each $1 \leq i \leq n$, if $x \Rightarrow_i^t y$ according to G', then $y \in (N \cup T)^*$. Therefore, we find $L_{t_c}(G') = L_{t_c}(G)$.

Now, it is sufficient to prove that the context-free grammar

$$G'' = (N', T, \bigcup_{i=1}^{n} P'_i, S)$$

generates the language $L(G')$. This is seen as follows. The inclusion $L(G') \subseteq L(G'')$ trivially holds, since one can mimic any derivation of G' by the context-free grammar G''. For $L(G'') \subseteq L(G')$ observe the following fact: consider an arbitrary derivation step $x \Rightarrow y$ according G'', where $x = z_1 A z_2$, $y = z_1 v z_2$, for some $z_1, z_2 \in (N \cup T)^*$, $A \in N$, and $A \to v$ is originally in P'_i. Note that x does not contain any symbol of the form $B^{(i)}$, $1 \leq i \leq n$. If y is a string over $N \cup T$, as well, then $x \Rightarrow^{t_c} y$ according to G' using the decomposition $x = x_0 x_1 x_2$ with $x_0 = z_1$, $x_1 = A$, $x_2 = z_2$, and the association ρ of components $\rho(1) = i$, keeping $\rho(0)$ and $\rho(2)$ undefined. If $|y|_{N^{(i)}} > 0$, then G'' has to continue the derivation earlier or later by replacing all the symbols from $N^{(i)}$ by some productions which originally come from P'_i, too. Since G'' is a context-free grammar, we can assume without loss of generality that (1) these replacements are performed immediately by G'', (2) those replacements are continued until a string y' is obtained with $|y'|_{N^{(i)}} = 0$, and (3) no symbols $B \notin N^{(i)}$ are replaced during this derivation $y \overset{*}{\Rightarrow} y'$. Then $x \Rightarrow_i^{t_c} y'$ holds by the same arguments as given above. Now, one easily proofs by induction that any derivation of G'' can be simulated by G' in the t_c-mode, and since G'' and G' have the same axiom, we have $L(G'') \subseteq L(G')$. □

It is known that the other competence based derivation protocol, namely the *sf*-mode, is more powerful than the *t*-mode. In [10] the equalities

$$\mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], sf).$$

have been shown. Moreover, it is known that $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac}) = \mathcal{L}(\mathrm{RE})$ and that $\mathcal{L}(\mathrm{ET0L}) \subset \mathcal{L}(\mathrm{P}, \mathrm{CF} - \lambda, \mathrm{ac}) \subset \mathcal{L}(\mathrm{CS})$, see, e.g., [6].

Concerning the cut-*sf*-mode, it is already seen from Example 2 that it is more powerful than the cut-*t*-mode, as well, since also non-context-free languages can be described. In fact, we even obtain the following result.

Lemma 6. $\mathcal{L}(\mathrm{RP}, \mathrm{CF}[-\lambda], \mathrm{ac}) \subseteq \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], sf_c)$.

Proof. First, let λ-productions be permitted. Let $G = (N, T, P, S, \Lambda, \sigma, \phi)$ be some recurrent programmed grammar with context-free core rules. From G we construct an CD grammar system $G' = (N', T, P_1, P_2, \ldots, P_n, S')$ such that $L_{sf_c}(G') = L(G)$ as follows. If $p \in \Lambda$ is a label to which $A \to v$ is associated in G, then let $N_p = \{p, p', p'', A_p, R_p\}$ be a set of new symbols and set $N' = \{S', R\} \cup N \cup \bigcup_{p \in \Lambda} N_p$, where R and S' are new symbols, again. Any derivation of G' is initiated by application of the component

$$P_{init} \quad = \quad \{\, S' \to pSR \mid p \in \Lambda \,\}.$$

With each production in P,

$$(p:\ A \to v,\ \{s_1, s_2, \ldots, s_k\},\ \{r_1, r_2, \ldots, r_m\}),$$

we associate the four new components

$$\begin{aligned}
P_{p,1} &= \{A \to A_p, p \to p', p' \to p'\} \cup \{\, B \to B \mid B \in N \,\} \\
P_{p,2} &= \{A_p \to v\} \cup \{\, p' \to s_i \mid 1 \le i \le k \,\} \cup \{\, B \to B \mid B \in N \,\} \\
P_{p,3} &= \{R \to R_p, R_p \to R_p, p \to p''\} \cup \{\, B \to B \mid B \in N \setminus \{A\} \,\} \\
P_{p,4} &= \{R_p \to R\} \cup \{\, p'' \to r_i, r_i \to r_i \mid 1 \le i \le m \,\} \cup \{\, B \to B \mid B \in N \setminus \{A\} \,\}
\end{aligned}$$

Finally, there is a terminating component

$$P_{term} \quad = \quad \{R \to \lambda\} \cup \{\, p \to \lambda \mid p \in \Lambda \,\}$$

The only component applicable to the axiom S' is P_{init}. Now, in every nonterminal sentential form, either R or R_p appears as right marker and one label symbol $p \in \Lambda$ or its primed or double primed version appears as left marker. Given a sentential form $p\alpha R$, $\alpha \in (N \cup T)^*$, the components $P_{p,1}$ and $P_{p,2}$ can be used in order to simulate the successful application of the production with label p of G, yielding a string $s_i\beta R$, $s_i \in \sigma(p)$ and $\beta \in (N \cup T)^*$, and $P_{p,3}$ and $P_{p,4}$ can simulate its application in appearance checking mode, yielding $r_i\alpha R$, $r_i \in \phi(p)$. For this, the complete sentential form is given to the components. When a string of the form pwR with $w \in T^*$ is obtained, the component P_{term} can be applied yielding w. This proves $L(G) \subseteq L_{sf_c}(G')$.

For the converse inclusion observe the following. A terminal word can only be obtained by applying P_{term} to a sentential form in $\Lambda T^*\{R\}$. Starting off with a sentential form $\alpha = p\beta R$, $p \in \Lambda$ and $\beta \in (N \cup T)^*$, neither the components $P_{q,2}$ nor $P_{q,4}$, $q \in \Lambda$, are applicable to any substring of α, since the presence of some symbol q or R_q is needed, respectively, in order to stop deriving. Therefore, we have to distinguish the following two cases:

1. Some $P_{q,3}$ becomes active first. If a component $P_{q,3}$ is applied to a substring α' of α, this component can become inactive only after the production $q \to q''$ has been used. Therefore, $q = p$ has to hold. Since p'' can be rewritten only with the help of $P_{p,4}$ and $P_{p,4}$ can stop deriving only by application of $R_p \to R$, the

symbol R_p must have been introduced when $P_{p,3}$ was active. Thus, $\alpha' = \alpha$ has to hold. In conclusion, $|\alpha|_A = 0$ since $P_{p,3}$ is not sf-competent on α otherwise. Furthermore, p'' must be replaced together with R_p during one and the same application of $P_{p,4}$. Therefore,

$$\alpha = p\beta R \Rightarrow^{sf_c}_{p,3} p''\beta R_p \Rightarrow^{sf_c}_{p,4} r_i\beta R\,,$$

$r_i \in \phi(p)$, is the only successful continuation of such derivation, simulating the application of the production with label p in appearance checking mode. Note that some $P_{q,1}$ may not be applied to some substring of $p''\beta R_p$ since it would introduce a symbol A_q such that $P_{p,4}$ cannot become active until A_q is rewritten again, but rewriting A_q can only be done with the help of $P_{q,2}$ which needs the presence of q' in order to finish its work in sf-mode of derivation.

2. Some $P_{q,1}$ becomes active first. Consider a derivation

$$\alpha \Rightarrow^{sf_c}_{q_1,1} \alpha_1 \Rightarrow^{sf_c}_{q_2,1} \alpha_2 \ldots \Rightarrow^{sf_c}_{q_\ell,1} \alpha_\ell$$

for some $\ell \geq 1$. Then we have $|\alpha_\ell|_{A_{q_i}} > 0$ for all $1 \leq i \leq \ell$. Let $\alpha_\ell = q_\ell u_0 A_{q_{i_1}} u_1 A_{q_{i_2}} u_2 \ldots A_{q_{i_\ell}} u_\ell$, where $u_j \in (N \cup T)^*$ for $1 \leq j \leq \ell$. The only possibility to get rid of a symbol A_{q_i} is the application of $P_{q_i,2}$ which can stop deriving on a sentential form only if q_i' is present. Therefore and since the application of some $P_{q,3}$ or $P_{q,4}$ must kept excluded from such derivations (see the arguments given for the first case), a string in $\Lambda(N \cup T)^*\{R\}$ can be obtained only if the following derivation is possible (due to the order of the symbols in α_ℓ):

$$\alpha_\ell \Rightarrow^{sf_c}_{q_{i_1},2} \gamma_1 \Rightarrow^{sf_c}_{q_{i_2},2} \gamma_2 \Rightarrow^{sf_c}_{q_{i_3},2} \ldots \Rightarrow^{sf_c}_{q_{i_\ell},2} \gamma_\ell\,,$$

where $q_{i_1} = q_\ell$ and $q_{i_{j+1}} \in \sigma(q_{i_j})$, for $1 \leq j < \ell$, hold. This simulates the successful applications of the corresponding productions of G.

Except from the fact that the two phases can be merged when components $P_{q,1}$ and when components $P_{q,2}$ are applied, no further terminating derivations are possible. Hence, $L_{sf_c}(G') \subseteq L(G)$, and the proof is finished for the case that λ-productions are allowed.

Let us remark that, even if the CD grammar system were forced, by some appropriate colouring of the symbols, to apply the component $P_{p,2}$ immediately after $P_{p,1}$ has been used, one would not be able to control how many symbols A are marked as A_p and then replaced with v. Therefore, the feature $p \in \sigma(p)$ for all $p \in \Lambda$ is an evident constraint, here.

For the λ-free case we argue as follows: By standard arguments the family $\mathcal{L}(\mathrm{CD}, \mathrm{CF} - \lambda, sf_c)$ is closed under union and embraces the finite languages. Let $L \subseteq T^*$ be in $\mathcal{L}(\mathrm{RP}, \mathrm{CF} - \lambda, \mathrm{ac})$, then

$$L = \bigcup_{a,b \in T} (a \cdot \delta_{a,b}(L) \cdot b) \cup (L \cap T^2) \cup (L \cap T) \cup (L \cap \{\lambda\}),$$

where $\delta_{a,b}(L) = \{ w \in T^+ \mid awb \in L \}$. Since L is in $\mathcal{L}(\mathrm{RP}, \mathrm{CF}-\lambda, \mathrm{ac})$, the language $\delta_{a,b}(L)$ is in $\mathcal{L}(\mathrm{RP}, \mathrm{CF}-\lambda, \mathrm{ac})$ due to closure properties of that family under left and right derivatives. The closure under derivatives is obvious, since $\mathcal{L}(\mathrm{RP}, \mathrm{CF}[-\lambda], \mathrm{ac})$ is a [full] AFL as shown in [8]. Thus, for the proof of the present assertion, it is sufficient to show that $\delta_{a,b,c}(L) \cdot abc$ is in $\mathcal{L}(\mathrm{CD}, \mathrm{CF}-\lambda, sf_c)$, provided that $\delta_{a,b,c}(L)$ is in $\mathcal{L}(\mathrm{RP}, \mathrm{CF}-\lambda, \mathrm{ac})$.

To this end, it is sufficient to exchange the terminating component P_{term} in the above construction with

$$Q_{term} \quad = \quad \{R \to b\} \cup \{ p \to a \mid p \in \Lambda \}$$

and the rest of the proof is given by the same arguments as in the first case. □

Any CD grammar system working in the sf_c-mode can be simulated by some Turing machine and, if λ-rules are forbidden, by some linear bounded automaton. Thus, together with the known facts about context-free CD grammar system working in the sf-mode, the following corollary is obtained.

Corollary 7. *1.* $\mathcal{L}(\mathrm{RP}, \mathrm{CF}, \mathrm{ac}) \subseteq \mathcal{L}(\mathrm{CD}, \mathrm{CF}, sf_c) \subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac}) = \mathcal{L}(\mathrm{RE})$

2. $\mathcal{L}(\mathrm{RP}, \mathrm{CF}-\lambda, \mathrm{ac}) \subseteq \mathcal{L}(\mathrm{CD}, \mathrm{CF}-\lambda, sf_c) \subseteq \mathcal{L}(\mathrm{CS})$

It is left open which of the inclusions in this corollary are strict. Finally we turn our attention to the combined modes. First let us summarize what is known for CD grammar systems working in non-cut modes. In case of the $(t \wedge \geq k)$-mode it was shown in [8] that

$$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], (t \wedge \geq 1)) \quad = \quad \mathcal{L}(\mathrm{ET0L})$$

and

$$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], (t \wedge \geq k)) \quad = \quad \mathcal{L}(\mathrm{RP}, \mathrm{CF}[-\lambda], \mathrm{ac}) \quad \text{if } k \geq 2,$$

and in [9] it was shown that

$$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda]),$$

for each $f \in \{ (t \wedge = k), (t \wedge \leq k) \mid k \geq 1 \}$, where $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$ denotes the family of languages of finite index generated by programmed context-free grammars without appearance checking. Loosely speaking, a grammar has finite index if the number of nonterminals in a derivation is bounded by a constant. For the definition of the *finite index* property see, e.g., [6].

Theorem 8. *If* $f \in \{ (t \wedge \leq k), (t \wedge = k), (t \wedge \geq k) \mid k \geq 1 \}$, *then*

$$\mathcal{L}(\mathrm{CF}) \subseteq \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

Proof. Let $G = (N, T, P, S)$ be a context-free grammar. In the remainder we restrict ourselves to the case $k = 1$. The result generalizes to arbitrary k, by using the prolongation technique, as elaborated in [8].

We construct a CD grammar system G' with nonterminals

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

where the unions are disjoint, with terminals T and axiom S. Define the homomorphism $h : (N \cup T)^* \to (N' \cup T)^*$ as follows: Let $h(A) = A'$ for $A \in N$ and $h(a) = a$ otherwise. Then we construct two production sets

$$P_1 = \{ A \to h(w) \mid (A \to w) \in P \} \quad \text{and} \quad P_2 = \{ A' \to A \mid A \in N \}.$$

This completes the description of the CD grammar system G'. It is easy to see that $L_{f_c}(G') = L(G)$, for $f \in \{(t \wedge \leq 1), (t \wedge = 1), (t \wedge \geq 1)\}$. This proves the claim. □

Corollary 9. *If* $f \in \{ (t \wedge = k), (t \wedge \geq k) \mid k \geq 2 \}$, *then*

$$\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

Proof. The inclusion follows from Theorem 8. The strictness follows from the CD grammar system specified in Example 1 when running in cut-$(t \wedge = 2)$ or cut-$(t \wedge \geq k)$, for $k \geq 2$, since the non-context-free language $\{ a^n b^n c^n \mid n \geq 1 \}$ is generated. In case of the cut-$(t \wedge = k)$-mode, for $k > 2$, the rules in the grammar of the above mentioned example have to be adapted accordingly with the prolongation technique in order to ensure that both nonterminals have to be presented during a $(t \wedge = k)$-derivation step. □

Since the $(t \wedge \geq 1)$-mode trivially coincides with the t-mode, we obtain the following corollary because of Theorem 5.

Corollary 10. $\mathcal{L}(\mathrm{CF}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], (t \wedge \geq 1)_c)$.

For the cut-$(t \wedge \geq k)$- and cut-$(t \wedge = k)$-mode in general we find the following situation. The below given theorem shows that with some cut-modes one can do even programmed context-free language of finite index.

Theorem 11. *If* $f \in \{ (t \wedge = k), (t \wedge \geq k) \mid k \geq 2 \}$, *then*

$$\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda]) \subset \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

Proof. We first show the inclusion. By a standard argument the involved CD grammar systems language family is closed under union and embraces the finite languages. Let $L \subseteq T^*$ be a language in $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$, then

$$L = \bigcup_{a \in T} (a \cdot \delta_a(L) \cap L) \cup (L \cap T) \cup (L \cap \{\lambda\}).$$

Since L is in $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$, language $\delta_a(L) = \{ w \in T^+ \mid aw \in L \}$ is in $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$ due to the closure properties of that family under derivatives. Thus, it is sufficient for the proof of the present assertion to show that $\{a\} \cdot \delta_a(L)$ is in $\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c)$, for $f \in \{ (t \wedge = k), (t \wedge \geq k) \mid k \geq 2 \}$, provided that $\delta_a(L)$ is a programmed context-free language of finite index. In the remainder we restrict ourselves to the case $k = 2$. The result generalizes to arbitrary k, by using the prolongation technique, as elaborated

Let $G = (N, T, P, S, \Lambda, \sigma, \phi)$ be a programmed grammar of finite index generating $\delta_a(L)$. Without loss of generality we assume that $N \cap \Lambda = \emptyset$, that for every rule $\Lambda(p) = (A \to w, \sigma, \emptyset)$ nonterminal A does not appear in w, and $p \notin \sigma(p)$. Moreover, due to the finite index restriction we may assume that every nonterminal appears at most once in any derivable sentential form.

We construct a CD grammar system G' with nonterminals

$$N' = N \cup \{\, p, p' \mid p \in \Lambda \,\} \cup \{S', S''\},$$

where the unions are disjoint, with terminals T and axiom S'. To start the derivation, we use S' as axiom and the component

$$P_{init} = \{S' \to S''\} \cup \{\, S'' \to pS \mid p \in \Lambda \,\}.$$

Then for each rule $\Lambda(p) = (A \to w, \sigma, \emptyset)$ we construct the component

$$P_p = \{\, p \to q \mid q \in \sigma(p) \,\} \cup \{A \to w\}.$$

Note that the $(t \wedge = 2)$-mode enforces the application of at exactly two rules, which must be the rule for the label p and the rule for the nonterminal A, because of the requirement that every nonterminal appears at most once in any derivable sentential form. Therefore the $(t \wedge = 2)_c$-derivation cuts the sentential form in such a way that only *one* production set will be applicable.

Finally, to terminate the derivation, process the component

$$P_{term} = \{\, p \to p', p' \to a \mid p \in \Lambda \,\}$$

is used. Observe, that after an application of P_{term} no other component, and in particular no component P_p, is successfully applicable to the sentential form because of the requirement on the appearance of nonterminals in any derivation. This completes the description of the CD grammar system G'.

Obviously, the constructed grammar system G' simulates the programmed context-free grammar G of finite index correctly and generates the language $\{a\} \cdot \delta_a(L)$. Moreover, note that the CD grammar system has λ-productions only if the programmed grammar has λ-productions.

The strictness immediately follows from Theorem 8 and the fact that the Dyck language (even over one pair of parentheses) cannot be generated by programmed context-free grammar of finite index as shown in [14]. □

Finally, we consider the cut-$(t \wedge = 1)$- and cut-$(t \wedge \leq k)$-mode, for $k \geq 1$. For all these modes we obtain a characterization of the context-free languages.

Theorem 12. *If* $f \in \{\, (t \wedge \leq k) \mid k \geq 1 \,\} \cup \{(t \wedge = 1)\}$, *then*

$$\mathcal{L}(\mathrm{CF}) = \mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c).$$

Proof. The inclusion from left to right was already shown in Theorem 8. Thus, it remains to consider the converse inclusion. Thus, let $G = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system working in f_c-mode, for the aforementioned f. First we

concentrate on f equals the $(t\wedge=1)$-mode. First, observe, that a cut-mode derivation can be sequentialized, such that only one production set is applicable to a sub-sentential form, i.e., the mapping ρ has domain size one. Now consider a derivation step: Let $x = x_0x_1x_2$ be a sentential form with $x_i \in (N \cup T)^*$, for $0 \leq i \leq 2$, and let $\rho(1) = i$, for some $1 \leq i \leq n$. Since a successful $(t \wedge =1)$-derivation on the sentential form x_1 has to apply exactly one rule of P_i and afterwards no rule of P_i is applicable anymore, we can assume that x_1 is an element of N, without changing the original successful derivation. This allows us to simulate the cut-$(t \wedge =1)$-derivation by a context-free derivation, choosing the appropriate rule of P_i. Thus, define the following context-free grammar $G = (N, T, P, S)$ with production set P containing the rules

$$P = \{\, A \to \alpha \mid A \Rightarrow_i^{(t\wedge=1)} \alpha \text{ for some } 1 \leq i \leq n \,\}.$$

By our previous investigation it is easy to see that the language generated by G is equivalent to the language generated by the CD grammar system working in cut-$(t \wedge =1)$-mode.

A similar reasoning applies for the cut-$(t \wedge \leq k)$-mode, since a $(t \wedge \leq k)$-mode derivation on a sentential form can be split or cut into several $(t \wedge \leq \ell)$-mode derivations, for $1 \leq \ell \leq k$, on nonterminals only. Thus, the production set of the context-free grammar has to be modified in the following way: Define

$$P = \{\, A \to \alpha \mid A \Rightarrow_i^{(t\wedge\leq\ell)} \alpha \text{ for some } 1 \leq \ell \leq k \text{ and } 1 \leq i \leq n \,\}.$$

Then the constructed context-free grammar is equivalent to the original CD grammar system working in cut-$(t \wedge \leq k)$-mode. This proves the stated claim. □

4 Conclusions

We examined the generative power of CD grammar systems when working in a new form of the derivation mode, the so called cut-mode. We summarize our results, comparing the classes $\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c)$ with $\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f)$, in Table 1. In most cases, the cut-f-mode turned out to be as most as powerful than the corresponding non-cut-mode, that is, the f-mode itself. For some modes, the generative capacity is not affected at all, but there are also some cases, where the power is reduced to that of context-free grammars, if parts of the sentential forms can be distributed to several components in one and the same derivation step. In particular, this is true for the t-mode of derivation, which describes the family of all ET0L languages in the non-cut variant. But exactly for this t-mode, the results are rather contesting the motivation of CD grammar systems from the AI point of view: Although CD grammar systems are a very natural and handy tool for describing non-context-free languages with context-free productions and they serve as sequential counterparts of tabled Lindenmayer systems, the re-interpretation of their properties in the framework of distributed problem solving is fairly problematic. In our approach, the cut-t-mode turns out to be useless, since it does not add to the power of context-free grammars. On the other hand, there seems to be something said for the $\geq k$- and $=k$-modes, $k \geq 2$, the generative capacity of which remains the same when they are driven in the cut variant. Also the sf- and some of the combined t-modes seem to

Derivation Mode f	$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f)$	$\mathcal{L}(\mathrm{CD}, \mathrm{CF}[-\lambda], f_c)$
$*$, $=1$, ≥ 1, $\leq k$, for $k \geq 1$	$\mathcal{L}(\mathrm{CF})$	$\mathcal{L}(\mathrm{CF})$
$=k$, $\geq k$, for $k \geq 2$	$\mathcal{L}(\mathrm{CF}) \subset \cdot \subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda])$	
t	$\mathcal{L}(\mathrm{ET0L})$	$\mathcal{L}(\mathrm{CF})$
sf	$\mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac})$	$\mathcal{L}(\mathrm{RP}, \mathrm{CF}[-\lambda], \mathrm{ac}) \subseteq \cdot \subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac})$
$(t \wedge \geq 1)$	$\mathcal{L}(\mathrm{ET0L})$	$\mathcal{L}(\mathrm{CF})$
$(t \wedge \geq k)$, for $k \geq 2$	$\mathcal{L}(\mathrm{RP}, \mathrm{CF}[-\lambda], \mathrm{ac})$	$\mathcal{L}(\mathrm{CF}) \subset \cdot$ $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda]) \subset \cdot$
$(t \wedge =1)$, $(t \wedge \leq k)$, for $k \geq 2$	$\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$	$\mathcal{L}(\mathrm{CF})$
$(t \wedge =k)$, for $k \geq 2$	$\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda])$	$\mathcal{L}(\mathrm{CF}) \subset \cdot$ $\mathcal{L}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda]) \subset \cdot$

Table 1: Generative capacity of CD grammars systems working in non-cut- and cut-mode derivations compared.

be of interest in that respect. Besides the combined derivation modes considered in this paper, there is another, external variant of hybrid CD grammar systems, e.g., see [12]. Those systems can also be considered in our setting. This is left for future research work.

References

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

[2] H. Bordihn, E. Csuhaj-Varjú, and J. Dassow. CD grammar systems versus L systems. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi Agent Systems*, pages 18–32. Gordon and Breach, 1999.

[3] H. Bordihn and M. Holzer. Grammar systems with negated conditions in their cooperation protocols. *Journal of Universal Computer Science* 6 (2000), 1165–1184.

[4] 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.

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

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

[7] J. Dassow, Gh. Păun, and G. Rozenberg. Grammar systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2, pages 155–213. Springer, 1997.

[8] H. Fernau, R. Freund, and M. Holzer. Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theoretical Computer Science*, 259(1–2):405–426.

[9] H. Fernau, M. Holzer, and R. Freund. Bounding resources in cooperating distributed grammar systems. In S. Bozapalidis, editor, *Proc. of the 3rd Int. Conf. Developments in Language Theory*, pages 261–272. Aristotle University of Thessaloniki, 1997.

[10] 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. Springer, 1978.

[11] P. H. Nii. Blackboard systems. In *The Handbook of Artificial Intelligence*, volume 4, pages 1–82, Addison-Wesley, 1989.

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

[13] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.

[14] B. Rozoy. The Dyck language $D_1'^*$ is not generated by any matrix grammar of finite index. *Information and Computation (formerly Information and Control)*, 74:64–89, 1987.

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

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 95 - 112, 2004.

CD Grammar Systems with $LL(k)$ Conditions*

Henning Bordihn

Institut für Informatik, Universität Potsdam
August-Bebel-Straße 89, D-14482 Potsdam, Germany
henning@cs.uni-potsdam.de

György Vaszil

Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
vaszil@sztaki.hu

Abstract

The paper extends the notion of context-free $LL(k)$ grammars to CD grammar systems working in the so-called $(=m)$ mode of derivation. The definition of the extended $LL(k)$ condition is followed by a short examination of the basic properties of $LL(k)$ CD grammar systems, and then an algorithm is presented which parses languages generated by CD grammar systems of this type in $O(n \cdot \log^2 n)$ time.

1 Introduction

The most investigated families of languages are the regular and the context-free ones. But, for many applications of formal languages, non-context-free aspects are needed. All programming languages, for example, in which variables must be declared before they can be used in the main program, involve the structure of the language $\{ ww \mid w \in \Sigma^* \}$, where Σ is some alphabet.

In [6], seven circumstances where context-free languages turn out to be insufficient are discussed, emerging in the field of programming languages, the language of logic, graph theory, development biology, economic modeling, folklore, and natural languages. The fact that there are non-context-free aspects in the syntax of spoken languages is exhaustively explored in the literature of linguistics, e.g., see [5, 9, 10]. This led to the concept of *mildly context-sensitive grammars* [9] providing some lower and upper bound conditions for families of languages which may be useful in linguistics. A grammar formalisms is said to be mildly context-sensitive if

*Research supported in part by the Intergovernmental S&T Cooperation Program of the Office of Research and Development Division of the Hungarian Ministry of Education and its German partner, the Federal Ministry of Education and Research (BMBF), under grant no. D-35/2000.

1. it contains (besides the context-free languages[1])

$$\begin{aligned} L_1 &= \{\, a^n b^n c^n \mid n \geq 1 \,\}, \\ L_2 &= \{\, wcw \mid w \in \{a,b\}^* \,\}, \\ L_3 &= \{\, a^m b^n c^m d^n \mid m,n \geq 1 \,\}, \end{aligned}$$

2. its generated languages can be parsed in polynomial time, and

3. it can generate only semilinear languages.

The next level in the Chomsky hierarchy, namely the context-sensitive grammars, are also not used in most applications since they are too powerful. For example, the fixed membership problem (i.e., the parsing problem) is **PSPACE**-complete and many other relevant decision problems are proved to be undecidable for context-sensitive grammars.

Therefore, a series of grammar formalisms has been introduced which are able to cover all the desired non-context-free aspects but maintain the nice properties of context-free grammars. Besides tree-adjoining grammars introduced in [9] and equivalent mildly context-sensitive grammars [16], the three most important sources of such language describing devices are grammars with parallel derivations (mainly Lindenmayer systems), grammars with regulated rewriting, and grammar systems; for a survey about these topics, see [14].

Unfortunately, most of those mechanisms lose too many positive properties of the context-free grammars, in particular the fixed membership problem becomes **NP**-complete in many cases. Even the known deterministic polynomial-time parsing algorithms for tree-adjoining and the other mildly context-sensitive grammars are of complexity $O(n^6)$, see, e.g., [10].

The aim of the present paper is to select one representative of those non-context-free devices and restrict it in a such a way that (1) at least the lower bound of mildly context-sensitive languages is met, that is, all the aforementioned languages L_1, L_2 and L_3 can be described, and (2) a parsing algorithm can be provided the computational complexity of which is as close to linear time as possible.

To this end, cooperating distributed grammar systems (CD grammar systems, for short) are considered here. The concept has its root in [11] aiming to generalize the notion of two-level substitution grammars to a multi-level concept, but the theory started with [3] introducing cooperating distributed grammar systems for describing, in terms of formal grammars and languages, communities of cooperating autonomous problem solving agents which use the blackboard model of problem solving. Later on, they were also considered as sequential counterparts of tabled Lindenmayer systems [1].

A cooperating distributed grammar system (CDGS, for short) consists of a finite set of (context-free) grammars, called components, performing derivation steps on a common sentential form in turns, according to some cooperation protocol. One natural cooperation protocol is the so-called $= m$-mode, for some $m \geq 1$, where a

[1]This requirement to cover all context-free languages has been weakened in more recent articles.

component, once started, has to perform exactly m derivation steps. The next component is nondeterministically chosen. In terms of distributed problem solving, the components correspond to the independent problem solving agents, the sentential form to the current state of the problem solving, and the generated language represents the set of problem solutions. For more on CD grammar systems see [4, 7]. It is known that this type of CD grammar systems generates all context-free languages and that it can describe matrix languages (without appearance checking).

One natural approach to fast parsers is to restrict the CD grammar systems to unambiguity. The present paper aims to do so by means of an $LL(k)$ condition for CD grammar systems. This yields a class of grammars which describe all context-free $LL(k)$-languages, the languages L_1, L_2, and L_3 above, and their languages can be parsed in time $O(n \cdot \log^2 n)$ time. Without further restriction, however, they do *not* define a new class of mildly context-sensitive languages since also non-semilinear languages can be generated.

It is worth mentioning here, that the logarithm in the time bound of the algorithm is squared only because we carefully count bit operations in which reading and writing a search index n takes $O(\log n)$ time. A uniform measurement of our algorithm would yield a time complexity of $O(n \cdot \log n)$.

2 Definitions

We assume the reader to be familiar with the basic notions of formal languages, as contained in [15]. In general, we have the following conventions: $\subseteq$ denotes inclusion, while $\subset$ denotes strict inclusion. The set of positive integers is denoted by $\mathbb{N}$ and the cardinality of a set M is denoted by $\#M$. By V^+ we denote the set of nonempty words over alphabet V; if the empty word ε is included, the we use the notation V^*. Set union and substraction is denoted by $\cup$ and $-$, respectively.

A context-free grammar is a four tuple $G = (N, T, P, S)$, where N and T are disjoint alphabets of nonterminals and terminals, respectively, $S \in N$ is the axiom, and P is a finite set of productions of the form $A \to u$, where $A \in N$ and $u \in (N \cup T)^*$.

Let V be some alphabet and $w \in V^*$. By $First_k(w)$ we denote the prefix of length k of w if $|w| \geq k$ or, otherwise, the string w itself.

Now we repeat the definition of a cooperating distributed grammar system, where we restrict ourselves (without further mentioning) to the case of context-free components.

A cooperating distributed grammar system (CDGS for short) of degree n is an $n+3$-tuple

$$G = (N, T, S, P_1, P_2, \ldots, P_n)\,,$$

where N and T are two disjoint alphabets, the alphabet of nonterminals and the alphabet of terminals, respectively, $S \in N$ is the axiom, and for $1 \leq i \leq n$, P_i is a finite set of context-free productions, i.e., productions of the form $A \to \alpha$, $A \in N$, $\alpha \in (N \cup T)^*$. The sets P_i are called components of the system G. For $1 \leq i \leq n$, we set $N_i = \{A \in N \mid A \to \alpha \in P_i\}$ and $T_i = (N \cup T) \setminus N_i$.

In the following, let x and y be sentential forms over $N \cup T$. A *direct derivation* according to some component P_i, $1 \leq i \leq n$, is defined as usual by $x \Longrightarrow_i y$ if and

only if $x = zAz'$, $y = z\alpha z'$, for some $z, z' \in (N \cup T)^*$ and $A \to \alpha \in P_i$. This relation is extended to an *m-step derivation* according to P_i, in symbols $x \overset{m}{\Longrightarrow}_i y$, in the natural way: $x \overset{m}{\Longrightarrow}_i y$ if and only if there are strings $x_0, x_1, \ldots, x_m$ over $N \cup T$ such that

$$x = x_0,\ y = x_m,\ \text{and } x_{j-1} \Longrightarrow_i x_j \text{ for } 1 \le j \le m\,.$$

Let $prod(x \overset{m}{\Longrightarrow}_i y)$ denote the set of production sequences which can be used in the m-step derivation $x \overset{m}{\Longrightarrow}_i y$. More precisely,

$$(A_0 \to \alpha_0, A_1 \to \alpha_1, \ldots, A_{m-1} \to \alpha_{m-1}) \in prod(x \overset{m}{\Longrightarrow}_i y)$$

if and only if there are strings $x_0, x_1, \ldots, x_m$ with $x_{j-1} = z_{j-1}A_{j-1}z'_{j-1}$ and $x_j = z_{j-1}\alpha_{j-1}z'_{j-1}$, and $A_{j-1} \to \alpha_{j-1} \in P_i$, for $1 \le j \le m$.

We write $x \overset{m}{\Longrightarrow}^* y$ if and only if $x = y$, or there are strings $x_0, x_1, \ldots, x_r$ over $N \cup T$, $r \ge 1$, such that

$$x = x_0,\ y = x_r,\ \text{and } x_{j-1} \overset{m}{\Longrightarrow}_{i_j} x_j\,,$$

for $1 \le i_j \le n$ for $1 \le j \le r$.

The language generated by G in the $(=m)$-mode of derivation is defined to be the set

$$L_{=m}(G) = \{\, w \in T^* \mid S \overset{m}{\Longrightarrow}^* w \,\}\,.$$

An m-step derivation $x \overset{m}{\Longrightarrow}_i y$ is referred to as *leftmost*, in symbols $x \underset{l}{\overset{m}{\Longrightarrow}}_i y$, if and only if the following condition is satisfied: If $x \overset{m}{\Longrightarrow}_i y$ is the derivation

$$x = x_0 \Longrightarrow_i x_1 \Longrightarrow_i x_2 \Longrightarrow_i \ldots \Longrightarrow_i x_m = y$$

for strings $x_0, x_1, x_2, \ldots, x_m \in (N \cup T)^*$, then we have

$$\begin{aligned} &x_0 = u_0A_0z_0,\ x_1 = u_0\alpha_0z_0,\\ &u_0 \in T^*,\ z_0 \in (N \cup T)^*,\ A_0 \to \alpha_0 \in P_i, \end{aligned}$$

and, for $2 \le j \le m$,

$$\begin{aligned} &x_{j-1} = u_{j-1}A_{j-1}z_{j-1},\ x_j = u_{j-1}\alpha_{j-1}z_{j-1},\\ &u_{j-1} \in T_i^*,\ z_{j-1} \in (N \cup T)^*,\ A_{j-1} \to \alpha_{j-1} \in P_i. \end{aligned}$$

That is, in the first step the leftmost occurrence of a symbol in N is replaced and in each of the following steps the leftmost occurrence of a symbol in N_i is replaced, each by some production in component P_i. This notion of leftmost derivation is the same as in [12] or as the notion of a *weakly leftmost* derivation in [8].

The relation $\underset{l}{\overset{m}{\Longrightarrow}}_i$ over $(N \cup T)^*$ is extended to $\underset{l}{\overset{m}{\Longrightarrow}}^*$ analogously to the extension of $\overset{m}{\Longrightarrow}_i$ to $\overset{m}{\Longrightarrow}^*$. The language generated by a CD grammar system G in the $(=m)$-mode of derivation working in this leftmost manner is defined by

$$L_{=m}(G\text{-left}) = \{\, w \in T^* \mid S \underset{l}{\overset{m}{\Longrightarrow}}^* w \,\}\,.$$

Next, we present the definition of an $LL(k)$ condition appropriate for CD grammar systems (in m-mode of derivation). It is adopted from the context-free case, for a definition see, e.g., [13, 15].

Definition 1 Let $G = (N, T, S, P_1, P_2, \dots P_n)$, $n \geq 1$, be a CDGS and let $k \geq 1$ and $m \geq 1$. We say G *satisfies the* $LL(k)$ *condition* in the $(=m)$-mode of derivation if the following holds: Let

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv \text{ and}$$

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy' \overset{m}{\underset{l}{\Longrightarrow}}{}_{i'} uz' \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv'$$

be two derivations in G, where $u, v, v' \in T^*$, $X \in N$, $y, y', z, z' \in (N \cup T)^*$, and $First_k(v) = First_k(v')$. Then $i = i'$ and $prod(uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz) = prod(uXy' \overset{m}{\underset{l}{\Longrightarrow}}{}_{i'} uz')$ is a singleton set.

The idea behind this concept is the following: Given a terminal word uv and a sentential form uXy, $X \in N$ and $y \in (N \cup T)^*$, which has been obtained from S by iterated leftmost m-step derivations, the first k letters of v allow to determine the next component and the sequence of rules of that component which is to be applied to uXy in order to derive uv by leftmost m-step derivations.

By $CD_n(=m)$ with $m, n \geq 1$ the family of languages which can be generated by a CD grammar system of degree n working in the m-mode of derivation is denoted. If we restrict the CD grammar systems of degree n to satisfy the $LL(k)$-condition in the $(=m)$-mode of derivation, then the families of languages obtained by leftmost derivations are denoted by $CD_nLL(k)(=m)$ with $k, m \geq 1$. If the number of components is not restricted, we write $CD_*(=m)$ and $CD_*LL(k)(=m)$, respectively. Finally, let $LL(k)$ denote the family of all context-free $LL(k)$ languages [15].

Example 1 Consider the CD grammar system

$$\Gamma = (\{S, S', S'', A, B, C, A', B', C'\}, \{a, b, c\}, S, P_1, P_2, P_3, P_4)$$

with

$$\begin{aligned}
P_1 &= \{S \to S', S' \to S'', S'' \to ABC\}, \\
P_2 &= \{A \to aA', B \to bB', C \to cC'\}, \\
P_3 &= \{A' \to A, B' \to B, C' \to C\}, \\
P_4 &= \{A \to a, B \to b, C \to c\}.
\end{aligned}$$

This system generates (by leftmost derivations) in the $(= 3)$ mode the language

$$L_1 = \{\, a^n b^n c^n \mid n \geq 1 \,\},$$

and as seen below, satisfies the $LL(2)$ condition, thus, $\{\, a^n b^n c^n \mid n \geq 1 \,\} \in CD_4LL(2)(= 3)$.

Considering a derivation

$$S \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz \overset{m}{\underset{l}{\Longrightarrow}}{}^{*} uv,$$

the pair $X \in N$ and $First_2(v)$ determines the component P_i and the unique production sequence in $prod(uXy \overset{m}{\underset{l}{\Longrightarrow}}{}_{i} uz)$ which are used, as indicated in the following table.

	aa	ab
S	P_1: $(S \to S', S' \to S'', S'' \to ABC)$	P_1: $(S \to S', S' \to S'', S'' \to ABC)$
A	P_2: $(A \to aA', B \to bB', C \to cC')$	P_4: $(A \to a, B \to b, C \to c)$
A'	P_3: $(A' \to A, B' \to B, C' \to C)$	P_3: $(A' \to A, B' \to B, C' \to C)$

It is an easy exercise to prove that $\{ a^n b^n c^n \mid n \geq 1 \} \in CD_4LL(2)(= 2)$ also holds. (Let A produce an a and b simultaneously.)

Example 2 Analogously, one can see that the CD grammar systems

$$\Gamma = (\{S, S', A, B, A', B'\}, \{a, b, c\}, S, P_1, P_2, \ldots, P_7)$$

with

$$\begin{aligned}
P_1 &= \{S \to S', S' \to c\}, \\
P_2 &= \{S \to S', S' \to AcB\}, \\
P_3 &= \{A \to aA', B \to aB'\}, \\
P_4 &= \{A \to bA', B \to bB'\}, \\
P_5 &= \{A' \to A, B' \to B\}, \\
P_6 &= \{A \to a, B \to a\}, \\
P_7 &= \{A \to b, B \to b\}.
\end{aligned}$$

and

$$\Gamma = (\{S, S', A, B, A', B'\}, \{a, b, c, d\}, S, P_1, P_2, \ldots, P_5)$$

with

$$\begin{aligned}
P_1 &= \{S \to S', S' \to AB\}, \\
P_2 &= \{A \to aA', B \to cB'\}, \\
P_3 &= \{A \to A'b, B \to B'd\}, \\
P_4 &= \{A' \to A, B' \to B\}, \\
P_5 &= \{A \to ab, B \to cd\}.
\end{aligned}$$

obey the $LL(2)$ condition in the =2-mode, generating (by leftmost derivations) the languages

$$L_2 = \{ wcw \mid w \in \{a, b\}^* \}$$

and

$$L_3 = \{ a^m b^n c^m d^n \mid m, n \geq 1 \},$$

respectively.

Hence, L_1, L_2, and L_3 from the definition of mild context-sensitivity are contained in $CD_*LL(2)(=2)$.

3 Properties of $CD_*LL(k)(=m)$

First, we present the following trivial hierarchies.

Lemma 1 *For any integers $k \geq 1$, $m \geq 1$, and $n \geq 1$, we have*

1. $CD_nLL(k)(=m) \subseteq CD_nLL(k+1)(=m)$,
2. $CD_*LL(k)(=m) \subseteq CD_*LL(k+1)(=m)$,
3. $CD_nLL(k)(=m) \subseteq CD_{n+1}LL(k)(=m)$.

Concerning the parameter m, one only knows from the non-restricted case that there exists a prime lattice structure-like hierarchy [4].

Next, we show that all context-free $LL(k)$ languages are in $CD_nLL(k)(=m)$ for some $n, m \geq 1$.

Theorem 2 *For all $k \geq 1$ and $m \geq 1$, $LL(k) \subseteq CD_*LL(k)(=m)$.*

Proof. Let $L \in LL(k)$ for some $k \geq 1$, and let $G = (N, T, P, S)$ be a context-free $LL(k)$ grammar with $L = L(G)$. Let the rules $r \in P$ be labelled by $1 \leq lab(r) \leq |P|$. For any integer $m \geq 1$, we construct a CD grammar system $\Gamma = (N', T, S, P_1, P_2, \ldots, P_n)$ satisfying the $LL(k)$ condition and generating L in the $=m$ mode of derivation as follows. The number of components of Γ is going to be $n = |P|$. Let

$$N' = N \cup \{ X^i \mid 1 \leq i \leq m-1,\ X \in N \},$$

and for $1 \leq i \leq |P|$, let

$$P_i = \{ X \to X^1, X^1 \to X^2, \ldots, X^{m-1} \to \alpha \mid i = lab(X \to \alpha) \}.$$

It is easy to see that a rewriting step $x \underset{l}{\overset{m}{\Longrightarrow}}_i y$ in Γ is possible if and only if $x \Rightarrow_i^l y$ is possible in G, where $\Rightarrow_i^l$ denotes a rewriting step on the leftmost nonterminal using rule r with $lab(r) = i$. □

Lemma 3 *There are non-semilinear languages in $CD_*LL(1)(=2)$.*

Proof. Consider the CD grammar system

$$G = (N,\ \{a, b, c, d, e, f\},\ P_1, P_2, \ldots, P_{13},\ S)$$

with $N = \{S, S', A, A', B, B', C, C', D, E, F, T, X\}$ and

$$\begin{array}{rcl@{\qquad}rcl}
P_1 & = & \{S \to S',\ S' \to AET\}, & & & \\
P_2 & = & \{A \to dA',\ E \to DD\}, & P_7 & = & \{B \to dB',\ D \to EE\}, \\
P_3 & = & \{A' \to X,\ X \to A\}, & P_8 & = & \{B' \to X,\ X \to B\}, \\
P_4 & = & \{A \to aX,\ X \to B\}, & P_9 & = & \{B \to aX,\ X \to A\}, \\
P_5 & = & \{A \to X,\ X \to C\}, & P_{10} & = & \{B \to X,\ X \to C\}, \\
P_6 & = & \{C \to eC',\ D \to b\}, & P_{11} & = & \{C \to fC',\ E \to b\}, \\
P_{12} & = & \{C' \to X,\ X \to C\}, & & & \\
P_{13} & = & \{C' \to c,\ D \to F,\ E \to F,\ T \to c\}. & & &
\end{array}$$

We are going to determine the language $L_{=2}(G\text{-left})$. Any leftmost derivation of G in the $=2$-mode starting off with the axiom must initially use component P_1 leading to AET. Then, the components P_2, P_3, P_4, P_7, P_8, and P_9 can be used in turns. In this phase, the number of occurrences of D's and E's can be increased. Note that this number can at most be doubled until a new symbol a will be introduced before a further increase is possible. Moreover, whenever one more D or E is introduced, then simultaneously a terminal d must emerge. This phase is finished by one application of either P_5 or P_{10} turning the leftmost nonterminal (A or B) to C. Now, all occurrences of nonterminals D and E can be terminated with the help of P_6, P_{11}, and P_{12}. Finally, the leftmost and the rightmost nonterminals, that is C' and T at this stage of the derivation, can be terminated by using P_{13}. Since also this terminating component must be applied in the leftmost way and F is a trap symbol, it is guaranteed that all occurrences of D and E have vanished before P_{13} can successfully be applied in the $=2$-mode. Therefore, in every non-terminal sentential form, either A, B, C (or its primed versions or X) is the leftmost occurring nonterminal, steering the selection of the components. Thus, the different phases of the derivation cannot be mixed.

Consequently, the non-semilinear language L is generated, where

$$L \subseteq K = \{fcbc\} \cup \{\, d^{i_1}ad^{i_2}a \dots d^{i_n}vcb^m c \mid n \geq 1,\, 0 \leq i_j \leq 2^j, \\ \text{for } 1 \leq j \leq n,\, m = 1 + \textstyle\sum_{j=1}^{n} i_j,\, v \in \{e,f\}^m \,\}\,.$$

Here, L is not equal to K only because the portion v has to obey some additional combinatorial properties which do not affect the non-semilinearity of the language. Since writing down these properties would decrease readability, they are omitted. On the other hand, the e's and f's are needed in order to make sure that G is $LL(k)$ in the $=2$-mode.

In fact, one can readily prove that G satisfies the $LL(1)$ condition. □

According to Lemma 1, we know that there exist non semilinear languages in $CD_nLL(k)(=2)$ for all $k \geq 1$ and all $n \geq 13$. By a simple prolongation technique, this is also true for any $=m$-mode of derivation, $m \geq 2$. One has to split the productions replacing the leftmost nonterminal to $m-1$ productions via new nonterminal symbols in each component. The restriction to leftmost derivations guarantees that these productions are consequently used such that the other production can be applied only once in any $=m$-step.

Corollary 4 *For any integers $k \geq 1$, $m \geq 2$, and $n \geq 13$, there are non-semilinear languages in $CD_nLL(k)(=m)$.*

4 Syntactic analysis of $CD_*LL(k)(=m)$

In this section we present a parser that is able to parse languages generated by CD grammar systems satisfying the $LL(k)$ condition in an efficient way. As we shall later see, its running time is $O(n \cdot \log^2 n)$ where n is the length of the input word.

Let G be a CD grammar system given as $G = (N, T, S, P_1, \dots, P_n)$ satisfying the $LL(k)$ condition in the $(=m)$-mode of derivation for some $m, k \geq 1$. First we present the notions we will use.

- A *production* is $p = X \to \alpha \in N \times (N \cup T)^*$ with $left(p) = X, right(p) = \alpha$.
- A *stack over* $\mathbb{N}$ is $st = x_j]x_{j-1}]\ldots]x_1]$, $x_i \in \mathbb{N}$, $1 \leq i \leq j$, with $top(st) = x_j$, $pop(st) = x_{j-1}]\ldots]x_1]$, and for some $y \in \mathbb{N}$, $push(y, st) = y]x_j]x_{j-1}]\ldots]x_1]$. The empty stack, $pop(x])$ for some $x \in \mathbb{N}$, is denoted by $\varepsilon]$.
- A *stack over* $N \cup T$ is $st = x_j]x_{j-1}]\ldots]x_1]$, $x_i \in N \cup T$, $1 \leq i \leq j$, with $top(st) = x_j$, $pop(st) = x_{j-1}]\ldots]x_1]$ and for some $y = y_1 \ldots y_m \in (N \cup T)^*$, $y_i \in N \cup T$, $1 \leq i \leq m$, $push(y, st) = y_1]\ldots]y_m]x_j]x_{j-1}]\ldots]x_1]$. The empty stack, $pop(x])$ for some $x \in N \cup T$, is denoted by $\varepsilon]$.
- A *production queue* is $pq = (p_1, p_2, \ldots, p_j)$, $p_i \in P_l$, $1 \leq l \leq n$, $1 \leq i \leq j$, with $first(pq) = p_1$, $butfirst(pq) = (p_2, \ldots, p_j)$.

The lookup table for the $LL(k)$ CD grammar system is given as $lookupTable \subseteq N \times T^k \times PQ$ where PQ denotes the set of all production queues consisting of m productions; it is a function which for a nonterminal $X \in N$ and a terminal word of length k, $y \in T^k$, returns a production queue $pq = lookupTable(X, y)$.

The parsing algorithm is given in Figure 1. It uses the variables

- $step$, $stepOfTopmost \in \mathbb{N}$, natural numbers,
- $mainStack$, a stack over $N \cup T$, the "main" stack of the parser,
- $stacksForN$, an l-tuple of stacks for natural numbers where $l = |N|$; it provides a stack over $\mathbb{N}$ for each nonterminal of the grammar system,
- $input \in T^*$, the string to be analyzed,
- $topmost \in N$, a nonterminal symbol,
- $pQueue$, a production queue as above,
- $pQueuesLeft \subseteq \mathbb{N} \times PQ$, where PQ denotes the set of all production queues of length at most m, that is, $pQueuesLeft$ is a set of pairs of the form $(i; pq)$ where i is an integer and pq is a production queue as above,
- $pToUse \in N \times (N \cup T)^*$, a production as above.

Example 3 In the following we demonstrate the work of the algorithm through an example.

Consider the CD grammar system

$$\Gamma = (\{S, A_1, A_2, A_3, A_4, A_5\}, \{a, b\}, S, P_1, P_2, P_3)$$

with

$$\begin{aligned} P_1 &= \{S \to A_1A_2A_1A_3, S \to bA_1A_2A_1A_3, A_2 \to b, A_3 \to A_4\}, \\ P_2 &= \{A_1 \to aA_2, A_4 \to A_5\}, \\ P_3 &= \{A_2 \to a, A_5 \to b\}, \end{aligned}$$

```
step ← 0
mainStack ← push(mainSt, S)
stacksForN(S) ← push(stacksForN(S), 0)
while mainStack is not empty and there is no ERROR do
  if top(mainStack) is a terminal symbol then
      if top(mainStack) coincides with the first symbol of input then
          mainStack ← pop(mainStack)
          input ← input without its first symbol
      else ERROR
else topmost ← top(mainStack)
      stepOfTopmost← top(stacksForN(topmost))
      stacksForN(topmost) ← pop(stacksForN(topmost))
      if there exist (i; pQueue) ∈ pQueuesLeft such that
      i ≥ stepOfTopmost, left(first(pQueue)) = topmost,
      and furthermore, if (i′; pQueue′) ∈ pQueuesLeft
      with left(first(pQueue′)) = topmost, then i < i′, then
          pQueuesLeft ← pQueuesLeft − {(i; pQueue)}
          pToUse ← first(pQueue)
          pQueue ← butfirst(pQueue)
          if pQueue is not empty then
              pQueuesLeft ← pQueuesLeft ∪ {(i; pQueue)}
          mainStack ← pop(mainStack)
          mainStack ← push(mainStack, right(pToUse))
          for each symbol X from right(pToUse) do
              if X ∈ N then
                  stacksForN(X) ← push(stacksForN(X), step)
      else step ← step + 1
          lookahead ← the next k symbols of input
          pQueue ← lookupTable(topmost, lookahead)
          if pQueue is empty then
              ERROR
          else pToUSe ← first(pQueue)
              pQueue ← butfirst(pQueue)
              if pQueue is not empty then
                  pQueuesLeft ← pQueuesLeft ∪ {(step, pQueue)}
              mainStack ← pop(mainStack)
              mainStack ← push(mainStack, right(pToUse)
              for each symbol X from right(pToUse) do
                  if X ∈ N then
                      stacksForN(X) ← push(stacksForN(X), step)
if there is no ERROR then successful termination
```

Figure 1: The parsing algorithm.

	a	b
S	P_1: $(S \to A_1A_2A_1A_3, A_2 \to b, A_3 \to A_4)$	P_1: $(S \to bA_1A_2A_1A_3, A_2 \to b, A_3 \to A_4)$
A_1	P_2: $(A_1 \to aA_2, A_1 \to aA_2, A_4 \to A_5)$	—
A_2	P_3: $(A_2 \to a, A_2 \to a, A_5 \to b)$	—

Figure 2: The lookup table for the grammar system of Example 3.

This system generates, in the $(= 3)$ mode, the finite language

$$L = \{aabaab, baabaab\},$$

and satisfies the $LL(1)$ condition. The lookup table for the parser is seen on Figure 2.

Let us see how the parser analyzes the string $aabaab \in L_{=3}(G\text{-left})$. This string is generated in three steps in the $(= 3)$-mode as follows.

$$S \overset{3}{\underset{l}{\Longrightarrow}}_1 A_1bA_1A_4 \overset{3}{\underset{l}{\Longrightarrow}}_2 aA_2baA_2A_5 \overset{3}{\underset{l}{\Longrightarrow}}_3 aabaab.$$

Now we will follow the work of the parser step-by-step, and describe its configuration by

$$(input, mainStack, step, stacksForN, pQueuesLeft)$$

where the variables are as described above. The value of $stacksForN$ will be denoted as $(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ where α_0 is the contents of $stacksForN(S)$, and for $i \in \{1, 2, 3, 4, 5\}$, x_i is the contents of $stacksForN(A_i)$.

The initial configuration of the parser is

$$(aabaab, S], 0, (0], \varepsilon], \ldots, \varepsilon]), \emptyset),$$

meaning that nothing is read from the input, the initial symbol, S, is placed in the main stack, the step counter is set to *zero*, the integer *zero* is placed in the stack $stacksForN(S)$ associated to the nonterminal S which indicates that it appeared in the main stack when the counter *step* had value *zero*, and the set of production queues waiting to be applied, $pQueuesLeft$, is empty.

The main stack is not empty, so the parser starts the execution of the **while** loop of the algorithm at line 4. The symbol on the top of the main stack is a nonterminal, so it jumps to line 10. Since $pQueuesLeft$, the set of production queues waiting for execution is empty, after popping the stack $stacksForN(S)$ associated to the symbol in the main stack, the parser proceeds with the instruction on line 24 by increasing the counter *step* and identifying the production queue to be applied with the help of the lookup table. At this point

$lookahead = a$,

$topmost = S,$
$pQueue = (S \to A_1A_2A_1A_3, A_2 \to b, A_3 \to A_4).$

The production to be used is the first production of $pQueue$,

$pToUse = S \to A_1A_2A_1A_3.$

Now the remaining part of $pQueue$ is stored in $pQueuesLeft$ indexed with *one*, the current value of the *step* counter, as a pair $(1; A_2 \to b, A_3 \to A_4)$. This indicates that the rules of this queue can be used on nonterminals that appeared in the main stack when the *step* counter had value *one* or less. Now the top of the main stack is replaced with the word on the right side of $pToUse$, the stack associated to S is emptied, and the value of *step* is placed into the stacks associated to the nonterminals appearing on the right side of the rule, $stacksForN(X)$, $X \in \{A_1, A_2, A_3\}$. The configuration of the parser is

$$(aabaab, A_1]A_2]A_1]A_3], 1, (\varepsilon], 1]1], 1], 1], \varepsilon], \varepsilon]), \{(1; A_2 \to b, A_3 \to A_4)\}).$$

Now the parser starts the execution of the **while** loop on line 4 again. Since the top of the main stack is a nonterminal, A_1, and since there is no production queue in $pQueuesLeft$ having A_1 on the left-hand side of its first rule, after popping $stacksForN(A_1)$, the stack associated with the topmost nonterminal, the parser continues with line 24 of the algorithm by increasing the counter *step*, and determining the production queue and the production to be used with the help of the lookup table, obtaining

$pQueue = (A_1 \to aA_2, A_1 \to aA_2, A_4 \to A_5),$
$pToUse = A_1 \to aA_2.$

After the application of the production $A_1 \to aA_2$ to the topmost nonterminal of the main stack, the parser is in the configuration

$$(aabaab, a]A_2]A_2]A_1]A_3], 2, (\varepsilon], 1], 2]1], 1], \varepsilon], \varepsilon]), \\ \{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5)\}),$$

and then the execution of the algorithm continues at line 4 again.

Since the top of the main stack is the same terminal as the first symbol of the input, the parser enters

$$(abaab, A_2]A_2]A_1]A_3], 2, (\varepsilon], 1], 2]1], 1], \varepsilon], \varepsilon]), \\ \{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5)\})$$

by popping the main stack and reading one letter of the input, then continues with line 4, and jumps to line 10 again.

Now the topmost nonterminal is A_2, and by popping *two* from the stack $stacksForN(A_2)$, it is clear that the production queue $(A_2 \to b, A_3 \to A_4)$ from $pQueuesLeft$ can not be used since it has index *one*. This means that the parser needs to turn to the lookup table again, obtaining

$pQueue = (A_2 \to a, A_2 \to a, A_5 \to b),$

$pToUSe = A_2 \to a.$

After the necessary replacements in the stacks and after updating the value of other variables, the parser enters

$$(abaab, a]A_2]A_1]A_3], 3, (\varepsilon], 1], 1], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_2 \to b, A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

Now after popping the main stack and reading one more symbol of the input, the parser continues at line 10 again. This time, the topmost nonterminal is A_2, and the integer popped from the corresponding stack, $stacksForN(A_2)$ is *one*, so the first production of the production queue $(A_2 \to b, A_3 \to A_4)$ stored in $pQueuesLeft$ with the same index can be used. Thus, the parser continues at line 14 of the algorithm setting

$$pToUse = A_2 \to b,$$
$$pQueuesLeft =$$
$$\{(1; A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}.$$

After replacing the topmost nonterminal of the main stack with the right side of $pToUse$, the parser enters

$$(baab, b]A_1]A_3], 3, (\varepsilon], 1], \varepsilon], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_3 \to A_4), (2; A_1 \to aA_2, A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

After popping the main stack and reading one more symbol of the input, the condition on line 13 is satisfied again, so the parser sets

$$pToUse = A_1 \to aA_2,$$
$$pQueuesLeft = \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\},$$

and then uses them, entering

$$(aab, a]A_2]A_3], 3, (\varepsilon], \varepsilon], 3], 1], \varepsilon], \varepsilon]),$$
$$\{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_2 \to a, A_5 \to b)\}).$$

Popping and reading again, then

$$pToUse = A_2 \to a,$$
$$pQueuesLeft = \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_5 \to b)\},$$

since the queue $(A_2 \to a, A_5 \to b)$ stored in $pQueuesLeft$ has index *three*, the same as the value obtained from the stack $stacksForN(A_2)$, so it can be used, producing

$$(ab, a]A_3], 3, (\varepsilon], \varepsilon], \varepsilon], 1], \varepsilon], \varepsilon]), \{(1; A_3 \to A_4), (2; A_4 \to A_5), (3; A_5 \to b)\}).$$

After the main stack is popped again and one more input symbol is read, the parser sets

$$pToUse = A_3 \to A_4,$$
$$pQueuesLeft = \{(2; A_4 \to A_5), (3; A_5 \to b)\},$$

and enters

$$(b, A_4], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], 1], \varepsilon]), \{(2; A_4 \to A_5), (3; A_5 \to b)\}).$$

The value *one* is placed in the stack $stacksForN(A_4)$ because the queue index of the rule $A_3 \to A_4$ was *one* which means that the application of the rule happens in step *one*, that is, A_4 appears in the first (= 3)-mode step of the generation of the input string.

The next configuration is

$$(b, A_5], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], 2]), \{(3; A_5 \to b)\}),$$

and then

$$(b, b], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon]), \emptyset),$$

after which the last input symbol is read and the main stack is once again popped, so the parser enters

$$(\varepsilon, \varepsilon], 3, (\varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon], \varepsilon]), \emptyset),$$

and since the main stack is empty, finishes its work at line 38 of the algorithm.

Theorem 5 *If a parser is constructed as above, based on a given CD grammar system satisfying the $LL(k)$ condition working in the $(=m)$ derivation mode, then it halts on every input word w over the terminal alphabet of the grammar system after a running time of $O(n \cdot \log^2 n)$ where n is the length of w.*

Proof. Let $G = (N, T, S, P_1, \ldots, P_s)$ be a CD grammar system satisfying the $LL(k)$ condition in the $(=m)$-mode of derivation. First we show that the parser constructed according to G halts on every input.

Assume that the parser does not halt on an input word $w \in T^*$. This means that it loops infinitely, and it can only do that if the instructions on the lines 10 – 37 are executed infinitely many times. To see this, notice that the body of the main **while** loop contains one **if-then-else** statement. Instructions of the **then** part read an input symbol, so they can not be repeated infinitely many times. This implies that the **else** part on lines 10 – 37 is repeated infinitely many times.

This part of the algorithm contains an **if-then-else** statement starting with line 13, the execution of the instructions of this part mean either the execution of the **then** part on lines 14 – 23, or the **else** part on lines 24 – 37. If lines 10 – 37 are executed infinitely many times, then there must be infinitely many such executions when no terminal symbol is written on the top of the main stack in line 20 or in line 34, which means that there is an infinite sequence of consecutive executions of lines 10 – 37 during which no terminal symbol is ever written on the top of the main stack.

Since each execution of the instructions of lines 14 – 23 removes one production from the production queues stored in $pQueuesLeft$, the instructions on lines 24 – 37 must be executed infinitely many times, or the parser cannot loop infinitely.

Because the lookahead never changes and because the number of nonterminal symbols is finite, there must be a sequence of instructions starting with lines 24 –

37, continuing with possibly several executions of lines 10 – 23 or lines 24 – 37, and then ending with lines 24 – 37 again, in such a way that the value of *topmost*, that is, the topmost nonterminal of the main stack, is the same at the first and at the last execution of lines 24 – 37.

Since the choice of the productions to be applied is based on the lookup table (line 26), the situation outlined above can only happen if the lookup table have certain properties which we describe below.

Let $X \in N$ and $y \in T^k$ be a row and a column index of the lookup table, and let $maxchain(X, y)$ denote the production queue with the following properties:

- $maxchain(X, y)$ is a prefix $(p_1, \ldots, p_l)$ of the corresponding entry of the lookup table, $lookupTable(X, y) = (p_1, \ldots, p_l, p_{l+1}, \ldots, p_m)$.
- If $maxchain(X, y)$ and $lookupTable(X, y)$ are as above, then $X \Rightarrow_{p_1} X_1 w_1$ and $X_i w_i \Rightarrow_{p_{i+1}} X_{i+1} w_{i+1}$, $X_i \in N$, $w_i \in (N \cup T)^*$, for each $1 \leq i \leq l-1$, and each p_i rewrites the leftmost nonterminal, that is, it is of the form $p_1 = X \to \alpha_1$, and $p_i = X_{i-1} \to \alpha_i$, $2 \leq i \leq l$, and furthermore,
- $maxchain(X, y)$ contains the maximal number of productions with the properties above, that is, $p_{l+1} = Z \to w$ where $Z \neq X_l$.

The parser may enter an infinite loop, if there exist a column of the lookup table, labelled with $y \in V^k$, such that

$$X \Rightarrow_{maxchain(X,y)} X_1 w_1 \Rightarrow_{maxchain(X_1,y)} X_2 w_2 \Rightarrow_{maxchain(X_2,y)} \cdots \\ \cdots \Rightarrow_{maxchain(X_l,y)} X_{l+1} w_{l+1} = X w_{l+1}$$

where $X, X_i \in N$, $w_i \in (N \cup T)^*$, $1 \leq i \leq l+1$, and $\Rightarrow_{maxchain(X,y)}$ denotes a leftmost derivation sequence using the rules of the production queue $maxchain(X, y)$.

Now we show that such a column cannot exist in the lookup table. If during a leftmost $(=m)$-mode derivation we encounter the nonterminal X as the leftmost nonterminal, and the production queue identified by X and the lookahead would be the queue in $lookupTable(X, y)$, then a successful application of the rules would lead to the choice of the queue $lookupTable(X_1, y)$, $lookupTable(X_2, y)$, and so on, until we would obtain X again as the leftmost nonterminal with the same lookahead, thus, the production queues identified by the leftmost nonterminal and the lookahead would never lead to a successful derivation which is a contradiction.

Now we show that given the input word $w \in T^*$, the parser halts after $O(n \cdot \log^2 n)$ number of steps where $n = |w|$. With similar arguments as above, we can show that the number of instructions executed without reading any input symbol is $O(1)$ which means that the running time of the parser is the length of the input multiplied by the time necessary to execute an instruction. All the instructions used in the algorithm can be executed in constant time, except the evaluation of the condition on line 13 and the assignments on line 14, 18, and 32 because they require the manipulation of the data stored in the set structure $pQueuesLeft$. The evaluation of line 13 requires a search, the assignments require the addition and the deletion of an element using a set where the number of stored elements can be as many as $O(n)$.

All of these operations, however, can be executed in $O(\log^2 n)$ time if we use balanced search trees, such as red-black trees for example, to store the elements of the set $pQueuesLeft$. (For more on balanced search trees and red-black trees in particular, see [2].) The implementation of the set $pQueuesLeft$ must consist of a red-black tree for each nonterminal $X \in N$ which stores the indexed production queues $(i; pq) \in pQueuesLeft$ with $X = left(first(pq))$ ordered by the index $i \in \mathbb{N}$. Having such a structure, the evaluation of the condition on line 13 can be realized by turning to the search tree associated to the nonterminal *topmost* to obtain the pair $(i; pQueue)$ where either $i = stepOfTopmost$, or if such index is not present, then i is the smallest available index with $i \geq stepOfTopmost$. To perform this search takes $O(\log n)$ comparisons since even in the worst case when the index is not present, it is enough to explore one path of the red-black tree leading from the root to one of the leaves, and the length of these paths, that is, the height of the tree is $O(\log n)$. To execute lines 14, 18, and 32, that is, to add or remove elements from the structure first requires a search to determine the appropriate tree and the location of the element in the tree, and then a constant number of elements need to be manipulated to insert or to remove the data. Since the number of trees used are finite, the number of necessary comparisons and data manipulations are $O(\log n)$. One comparison or one data manipulating step, however, also requires $O(\log n)$ time, since the integers used to index the production queues, that is, the keys used to index the nodes of the search tree, might be as large as n, so their representation can be as long as $\log n$ which means that comparing, reading or writing them requires $O(\log n)$ elementary computation steps. This gives a total running time of $O(n \cdot \log^2 n)$ where $n = |w|$, the length of the input word. □

5 Conclusion

Cooperating distributed grammar systems working in $= m$-mode of derivation have been restricted in a way such that, on the one hand, they maintain enough power in order to generate all context-free $LL(k)$ languages, the languages L_1, L_2 and L_3 of the concept of mildly context-sensitive grammars and even some non-semilinear language, but, on the other hand, there is an efficient parsing algorithm of $O(n \cdot \log^2 n)$ time complexity. The focus in this paper was on the development of the concept of an $LL(k)$ condition which is appropriate for those systems, and of the parsing algorithm. The corresponding families of languages $(CD_nLL(k)(= m))$ need further investigations. The future research should investigate, among others, the following problems:

- Which of the inclusions given in Lemma 1 are strict?
- Is it decidable whether a given CD grammar system is $LL(k)$, for a given k or for any k?

Moreover, one could extend the research to other derivation modes. Finally, other restrictions like an appropriate $LR(k)$ condition can be taken into consideration.

References

[1] Bordihn, H., Csuhaj-Varjú, E., Dassow, J.: CD grammar systems versus L systems, in: Gh. Păun, A. Salomaa, (eds.), *Grammatical Models of Multi-Agent Systems*, Gordon and Breach, 1999, 18–32.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill Book Company, 1990.

[3] Csuhaj-Varjú, E., Dassow, J.: On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics* EIK, **26** (1990), 49–63.

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

[5] Ch. Culy, The complexity of the vocabulary of Bambara. *Ling. and Philosophy* **8** (1985), 345–351.

[6] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory.* Springer, 1989.

[7] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems. Chapter 4. *Handbook of Formal Languages*, in: G. Rozenberg and A. Salomaa (eds.), Springer, Berlin, 1997, 155-213.

[8] J. Dassow, V. Mitrana, On the leftmost derivation in cooperating grammar systems. *Rev. Roumaine Math. Pures Appl.*, **43** (1998), 361-374.

[9] A. K. Joshi, How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars, in: D. Dowty, L. Karttunen, and A. Zwicky (eds.), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, Cambridge University Press, New York, 1985.

[10] A. K. Joshi, Y. Schabes, Tree-adjoining grammars, in: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, Springer, Berlin, 1997, 69–123.

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

[12] V. Mitrana, Parsability approaches in CD grammar systems, in: R. Freund and A. Kelemenová (eds.), *Proceedings of the International Workshop Grammar Systems 2000*, Silesian University at Opava, 2000, 165–185.

[13] D. J. Rosenkrantz, R. E. Stearns, Properties of deterministic top-down grammars, *Information and Control*, **17** (1970), 226–256.

[14] G. Rozenberg, A. Salomaa, *Handbook of Formal Languages*, Springer, Berlin, 1997.

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

[16] K. Vijay-Shanker, D. J. Weir, The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* **87** (1994), 511–546.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil, MTA SZTAKI, Budapest, pages 113 - 125, 2004.

On the Time, Space and Communication Complexity of Cooperating Distributed Grammar Systems

Liliana Cojocaru

Rovira i Virgili University of Tarragona
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
liliana.cojocaru@estudiants.urv.es

Abstract

In this paper we deal with trade-offs between *time, space,* and *communication complexity* of languages generated by Cooperating Distributed Grammar Systems (henceforth CDGS) with regular, linear and context-free components. We propose two types of communication structures. The first structure is determined by the *communication graph* of the CDGS, i.e, a directed graph in which the vertices are labeled by the CDGS components and the directed edges correspond to pairs of grammars (G_a, G_b), $a \neq b$, that communicate with each other. The communication is done through those nonterminals that appear on the right side of a production of G_a and on the left side of a production of G_b, according to the protocol of cooperation used by the system. We will refer to these nonterminals as *communicational nonterminals.* The second structure is given by the communicational protocol tree attached to the generated language, i.e. the derivation tree of a special kind of *Szilard language* introduced in this paper, called *communicational Szilard language.* A *communication complexity* measure, i.e. how many times the system components communicate with each other using a minimal number of communicational nonterminals, is defined and studied depending on modes of derivation, weak and strong fairness conditions. We found that the communication complexity of weakly and strongly q-fair languages in the case of grammar systems with regular or linear components is linear. These languages can be accepted by a nondeterministic multitape Turing machine in linear space and linear time. In the case of languages generated by CDGS with context-free components we show that the communication complexity varies from linear to logarithmic. The space required by a nondeterministic multitape Turing machine to accept these languages equals to the communication complexity, while the time is linear in all the cases.

1 Introduction

Communication complexity is one of the youngest branches of complexity theory. It has been inspired from the inter-processor communication, in which the input, viewed as sequences of messages distributed among different parts of a system, has to be split in different partitions in order to allow an optimal communication. It has been introduced in 1979 by Yao [17], and it studies the communication exchanged during a computational process by *minimizing the amount of information* exchanged

between the system components. The flow of information (the number of exchanged bits) is measured by ignoring the other cost, such as time and space.

This paper is devoted to the communication complexity of Cooperating Distributed Grammar Systems, with regular, linear and context-free components, but concerns also other computational resources used by the system, such as time and space. We deal with trade-offs between this three measures in order to control the generative process underlined by CDGS.

Investigations related to the communication complexity of distributed grammar systems have been done so far in several papers, e.g. [9], [10], [11], [12], [13]. In these papers the authors focus on the communication complexity of *Parallel Communicating Grammar Systems* (henceforth *PCGS*). They have considered two kinds of communication complexity measures. The first one is the *communication structure of PCGS*, i.e. the shape of the communication graph, consisting of directed communication links between the grammars, while the second one is a *communication complexity measure*, i.e. the number of exchanged messages during the computational process.

For the case of CDGS we propose two types of communication structures. The *first structure* is determined by the *communication graph* of CDGS, which is a directed graph where the vertices are labeled by the CDGS components and the directed edges correspond to pairs (G_a, G_b), $a \neq b$, of grammars that communicate with each other. The communication is done through those nonterminals that appear on the right side of a production of G_a and on the left side of a production of G_b, according to the protocol of cooperation used by the system. We refer to these nonterminals as *communicational nonterminals.* The *second structure* is a protocol tree determined by the interconnection between the system components, i.e. the way in which they bring consecutive contributions on the sentential form during the language generation process. We define a new kind of language, called *communicational Szilard language.* The derivation tree of a certain word γ_w^c from this language is the *communicational protocol tree* attached to the corresponding word w from the language generated by the system, for which γ_w^c is the *communicational control word* of w. A *communication complexity measure*, i.e., how many times the system components communicate with each other using a minimal number of communicational nonterminals, is defined and studied depending on modes of derivation, weak and strong fairness conditions.

2 Prerequisites

Grammar systems are sets of grammars that function together according to a specified protocol of cooperation. In CDGS all the components have a common axiom, all grammars have the same working tape, each of them making their own contribution on the common sentential form. At each moment only one grammar is active. Which component of the system is active at a given moment, and when a grammar stops to be active, is decided by the protocol of cooperation. This protocol consists in stop conditions such as modes of derivation (how many times a rewriting rule of the same component can be applied), in weak fairness conditions

(each component has to be activated almost the same number of times) or in strong fairness conditions (each component has to be activated almost the same number of times, by taking into account the number of internal productions that are applied for each grammar). CDGS simulates the blackboard model of problem solving, in which the blackboard is the common working tape, and the components $G_1, ..., G_r$ are the knowledge sources (agents, processors, etc.). They have been introduced and studied in [2], [3], [4] and [8], with forerunner related papers [14] and [15]. Formally a CDGS is defined as follows:

Definition 1 A **Cooperating Distributed Grammar System** of degree r, $r \geq 1$ is a construct of the form: $\Gamma = (N, T, S, P_1, \ldots, P_r)$, (1)
where the set N and T are disjoint finite alphabets, the *nonterminal* and the *terminal* alphabet, respectively. $S \in N$ is the system axiom, and $P_1, P_2, \ldots, P_r$ are finite sets of rewriting rules over $N \cup T$.

(1) can be rewritten equivalently: $\Gamma = (N, T, S, G_1, \ldots, G_r)$, (1')
in which $G_i = (N, T, S, P_i)$, for all i, $1 \leq i \leq r$, are Chomsky grammars, called the *components* of Γ. For $X \in \{REG, LIN, CF\}$ we denote by CDGS$_r$X, $r \geq 1$, CD grammar systems with r components, that have regular, linear and context-free components, respectively. The language generated by these systems depends on the way in which the internal rules of each component bring their own contribution on the sentential form. This can be done with respect to several modes of derivation, defined below:

Definition 2 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS, $x, y \in (N \cup T)^*$, and $i \in \{1, ..., r\}$. The **terminating derivation** (denoted by $\Rightarrow^t_{P_i}$), the **k-steps derivation** (denoted by $\Rightarrow^{=k}_{P_i}$), **at most k-steps derivation** (denoted by $\Rightarrow^{\leq k}_{P_i}$), **at least k-steps derivation** (denoted by $\Rightarrow^{\geq k}_{P_i}$), and the ***-mode of derivation** (denoted by $\Rightarrow^*_{P_i}$), represent modes of derivations that allow for each component P_i to consecutively activate: as many rules as possible, exactly k rules, at most k rules but at least one rule, at least k rules, and arbitrarily many rules, respectively.

Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS and $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$.

Definition 3 The language generated by Γ in *f-mode*, $f \in M$ is defined as:
$L_f(\Gamma) = \{w \in T^* | S = w_0 \Rightarrow^f_{P_{i_1}} ... \Rightarrow^f_{P_{i_m}} w_q = w, m \geq 1, 1 \leq i_j \leq r, 1 \leq j \leq m\}$.

For $X \in \{REG, LIN, CF\}$, $f \in M$ we denote by $CD_rX(f)$, $r \geq 1$, the family of languages generated by CDGS with r components, that have only regular, linear, and context-free rules activated in the f-mode of derivation.

Definition 4 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$, be a CDGS. The **control word** of w, with respect to the system components that have been applied in f-mode, $f \in M$, for a terminal derivation: $S = w_0 \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2 ... \Rightarrow^f_{P_{i_m}} w_q = w$ is defined as $\gamma_w = P_{i_1} P_{i_2} ... P_{i_m}$. The **Szilard language** associated to the derivation in the f-mode, in Γ is: $Sz(\Gamma, f) = \{\gamma_w | w \in L_f(\Gamma), f \in M\}$.

We denote by $SZ(f)$ the *family of Szilard languages* $Sz(\Gamma, f)$ for any grammar system Γ, in the f-mode of derivation. For more properties of these languages the reader is referred to [7].

Definition 5 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$, be a CDGS. The **communicational control word** of w, that is a control word built with respect to the communicational nonterminals used during a terminal derivation of the form $S = w_0 \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2 ... \Rightarrow^f_{P_{i_m}} w_q = w$, in f-mode, $f \in M$, is defined as $\gamma^c_w = P^{n_1}_{i_1} P^{n_2}_{i_2} ... P^{n_q}_{i_m}$, where n_j, is the *number of communicational nonterminals* rewritten during the application of rules of the component P_{i_j}, $1 \leq j \leq m$. The **communicational Szilard language** associated to a terminal derivation in the f-mode, in Γ is defined as:

$$Szc(\Gamma, f) = \{\gamma^c_w | w \in L_f(\Gamma), f \in M\}.$$

We denote by $SZC(f)$ the *family of communicational Szilard languages* $Szc(\Gamma, f)$ for any grammar system Γ, in f-mode of derivation. Note that in the case of grammar systems with regular and linear components the languages $Sz(\Gamma, f)$ and $Szc(\Gamma, f)$ are equal. They can be different only in the case of grammar systems that contains at least one non-linear rule.

Besides modes of derivation other restrictions that control the generative process are given by fairness conditions. Informally, these conditions require that all components of the system have approximately the same contribution on the common sentential form. They have been introduced in [6], in order to control and to increase the generative capacity of grammar systems. Formally they are defined as follows:

Definition 6 Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$, be a CDGS, and

$$\text{D: } S = w_0 \Rightarrow^{=n_1}_{P_{i_1}} w_1 \Rightarrow^{=n_2}_{P_{i_2}} w_2 ... \Rightarrow^{=n_q}_{P_{i_m}} w_q = w$$

be a derivation in f-mode, where P_{i_j} performs n_j steps, $1 \leq j \leq m$. For any $1 \leq p \leq r$, we set

$$\psi_D(p) = \sum_{i_j = p} 1 \quad \text{and} \quad \varphi_D(p) = \sum_{i_j = p} n_j \tag{2}$$

- *the weak maximal difference between the contribution of two components involved in the derivation D* is defined as:

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

- *the strong maximal difference between the contribution of two components* is:

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

Let $u \in \{w, s\}$, $x \in (N \cup T)^*$, $f \in M$ and

$$du(x, f) = min\{du(D)| \text{ where D is a derivation of } x \text{ in } f\text{-mode}\},$$

for a fixed natural number $q \geq 0$,

- the **weakly q-fair** language generated by Γ in the f-mode is defined as:

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

- the **strongly q-fair** language generated by Γ in the f-mode as:

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

For $X \in \{REG, LIN, CF\}$ and $f \in M$, $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$ we denote by $CD_rX(f, w - q)$ and $CD_rX(f, s - q)$, $r \geq 1$, the family of weakly and strongly q-fair languages, respectively, generated by CDGS with r components, that have regular, linear, and context-free components in the f-mode of derivation.

3 The Amount of Communication

In this section, two communication structures are proposed in order to investigate the communicational process of distributed generation of languages for the case of CDGS. The first structure is given by the *communication graph* of the CDGS, while the second structure is given by the *protocol of collaboration* between the system components, and it is strictly related to the structure of the communicational Szilard language associated to a language generated by CDGS. We call this structure the *communicational protocol tree* underlied by the grammar system. Another measure deals with the number of communicational steps spent during the computational process. We call it *communication complexity*. In what follows we describe how these complexity measures work together, and how they can be used to characterize the communicational process of CDGS.

3.1 Measures of Communication

Let $\Gamma = (N, T, S, G_1, \ldots, G_r)$ be a CDGS and $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$.

Definition 7 The **communication graph** of a CDGS crossed in the f-mode of derivation, $f \in M$, is a directed graph in which the vertices are labeled by the CDGS components that communicate with each other. Each directed edge, from a node labeled by G_a to another node labeled by G_b, $a \neq b$, corresponds to a communication step from the component G_a to the component G_b, done during the derivational process, i.e., there exists at least one nonterminal that appears on the right side of a production from G_a and of the left side of a production from G_b, rewritten at least one time during the derivational process in the f-mode.

Definition 8 The **communicational protocol tree** attached to a word $w \in L_f(\Gamma)$, $f \in M$ is the derivation tree attached to the communicational control word of w, i.e. γ_w^c, in the f-mode.

Note that the number of sons of a given node in the communicational protocol tree depends on the type of the rule through which the communication is performed. In the case of regular or linear rules, a grammar G_a communicates with another grammar G_b through only one nonterminal, so that the corresponding protocol tree will be a simple tree (each node has only one son). In the case of non-linear rules the number of sons equals the number of communicational nonterminals from the right side of the rule. Consequently, the shape of the communicational protocol tree depends not only on modes of derivation, but also on the number of communicational nonterminals that exist on the right side of the same production. Therefore, it might exist grammar systems with non-linear rules for which only one nonterminal from the right side of each production is a communicational one. In this case the protocol tree will be a simple tree, too. The communicational protocol tree is not a simple tree, only in the case of CDGS for which there exists at least one non-linear rule that has at least two communicational nonterminals at the right side of it.

The *communication complexity* measure represents the number of communications between different components, during the generative process, by using a minimal number of communicational nonterminals in a specified mode of derivation. This

communication measure represents in fact how many times, in a terminal derivation, different components can be applied (this fact can be restricted only by the f-mode of derivation or fairness conditions). Due to the above observation the communication complexity is a function of the length of the generated word depending on $\psi_D(p)$, $1 \leq p \leq r$, introduced in (2). Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$, be a CDGS, and D a derivation in Γ, such that $D : S \Rightarrow^f_{P_{i_1}} w_1 \Rightarrow^f_{P_{i_2}} w_2 ... \Rightarrow^f_{P_{i_m}} w_q = w$.

Definition 9 We denote by $Com(D) = \sum_{p=1}^{r} \psi_D(p)$, where $\psi_D(p) = \sum_{i_j=p} 1$, the number of communication steps used during the derivation D.
The **communication complexity of a word** w, $w \in L_f(\Gamma)$ is defined as:

$$Com(w, \Gamma) = min\{Com(D) | D : S \Rightarrow^* w\}.$$

The **communication complexity of** Γ over all words of length n is:

$$Com_\Gamma(n) = sup\{Com(w, \Gamma) |, w \in L_f(\Gamma), |w| = n\}.$$

The **class of languages that can be generated within communication f** by a CDGS, is defined as: $COM(g) = \bigcup_\Gamma \{L_f(\Gamma) | Com_\Gamma = O(g)\}$.

To be observed that in the case of CDGS with regular or linear components the control language and the communicational control language are equal. Furthermore, due to the fact that the height of the communicational protocol tree of a certain word w is equal with the length of the derivation of the control word associated to w, Definition 9 of the communication complexity can be equivalently redefined as follows.

Definition 10 The number of communication steps, i.e $Com(D)$, used during a derivation D of a particular word w is the height of the communicational protocol tree attached to the communicational control word.

The **communication complexity of a word w**, $w \in L_f(\Gamma)$, i.e $Com(w, \Gamma)$, is the minimum of the heights over all communicational protocol trees attached to each communicational control words of w.

The **communication complexity of** Γ over all words of length n, i.e $Com_\Gamma(n)$, is the supremum of the heights over all minimal communicational protocol trees.

Let Γ be a CDGS, and D be a (minimal) terminal derivation of w, where $w \in L_f(\Gamma)$, $f \in M$, $M = \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$. We denote by $|\gamma_w(D)|$ and $|\gamma_w^c(D)|$ the length of the derivation of the control word, $|\gamma_w|$, and of the communicational control word, $|\gamma_w^c|$, associated to w, respectively. Then, the next theorem holds.

Theorem 1 For each grammar system Γ, that has only useful components[1] and $w \in L_f(\Gamma)$, we have:
1. $|\gamma_w(D)| = Com(|w|)$.
2. There exist two positive constants a and b, such that $a|\gamma_w^c(D)| \leq |w| \leq b|\gamma_w^c(D)|$.

Proof. The first claim is a direct consequence of Definitions 9 and 10. To prove the second claim we consider firstly the case of regular and linear components. In this

[1] That is each component brings contributions on the sentential form, directly through terminal symbols (in the case of regular or linear rules), or indirectly through non-terminal symbols (in the case of context-free components).

case at each step between two consecutive communications at least one terminal symbol is brought into the sentential form. Therefore, the generated word will contain at least the number of communication steps performed during the generative process. In this case we have $a = 1$, and b equals the maximum number of the terminal symbols brought into the sentential form by each rule of the system.

In the case of grammar systems with (non-linear) context-free components, let us consider γ_w^c of the form $\gamma_w^c = P_{i_1}^{n_1} P_{i_2}^{n_2} ... P_{i_k}^{n_k}$. The leaves in the associated protocol tree correspond to those components that contain rules that have on their right side only terminals. They contribute in the sentential form with substrings of w. The worst case happens when no terminal symbol is brought into the sentential form between any two consecutive communication steps. Therefore, the number of the leaves, i.e. n_k, is less than or equal to the length of w, and the length of w cannot be more then n_k multiplied by a constant c. So that we have

$$n_k \leq |w| \leq cn_k \leq c_{max}|\gamma_w^c(D)|. \tag{3}$$

On the other hand there are situations when between each two consecutive communication steps each component contributes in the sentential form with a constant number of terminals. Each communicational nonterminal might bring its own contribution on the sentential form, too. That is why we also have

$$c_{min}|\gamma_w^c(D)| = c_{min}(n_1 + n_2 + ... + n_k) \leq c_1 n_1 + c_2 n_2 + ... + c_k n_k \leq |w| \tag{4}$$

From (3) and (4) we have $a = c_{min}$ and $b = c_{max}$. □

Due to the above result the communication complexity of CDGS is strictly related not only to modes of derivation or fairness conditions but also on the types of the rules of the system components, and on the number of non-terminals of a non-linear rule that are communicational non-terminals. In the next subsection, we state several results that concern the time, space and the communication complexity of CDGS with regular, linear and context-free components.

3.2 CDGS with Regular and Linear Components

Theorem 2 For each grammar system $CDGS_r X$, with $X \in \{REG, LIN\}$ and $r \geq 2$, there exists a $CDGS$ with only one component that will generate the same language, and vice-versa, independently of modes of derivation.

Corollary 1 $CD_* X = X$, for $X \in \{REG, LIN\}$, independently of modes of derivation.

Corollary 2
$SZC(CDGR_* X, f) = SZ(CDGR_* X, f) = REG$, for $X \in \{REG, LIN\}$, and $f \in \{*, t\} \cup \{\geq k, = k, \leq k | k \geq 1\}$.

Corollary 3 The communication complexity of $CD_* X(f)$, $X \in \{REG, LIN\}$, $f \in \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$ is 0.

Corollary 4 $LIN \subseteq COM(0)$.

It is well known that the communication complexity divides languages in small complexity classes. The above results show that the communicational process of

CDGS is a lazy one. So that the process of communication in these system is not so powerful as it has been proved to be for the case of PCGS, where several hierarchies of very (small) complexity classes have been found. Due to the fact that fairness conditions increase the generative power of a grammar system, the above theorem and corollaries do not hold for the case of q-fair languages. A CDGS with arbitrary number of components cannot be "compressed" into a single grammar that generates the same q-fair language, by preserving the mode of derivation, too. Even for these types of languages in the case of a constant communication, the class of weakly q-fair languages generated by CDGS with regular or linear components coincide with the languages generated by the same grammar without any weak fairness condition, so that due to Corollary1 we have:

Corollary 5 $CD_{r,c}X(f, w-q) \subseteq COM(0)$ and $CD_{r,c}X(f, s-q) \subseteq COM(c)$ where $X \in \{REG, LIN\}$ and c is the constant number of communicational steps performed during the derivation.

Corollary 6 The communication complexity of weakly/strongly q-fair languages, $L_f(\Gamma, w-q)/L_f(\Gamma, s-q)$, $f \in \{t\} \cup \{\leq k, = k, \geq k | k \geq 1\}$, generated by $CDGS_*X$, $X \in \{REG, LIN\}$, for which the communication graph is a tree or a dag is 0/constant.

Nevertheless the above results do not hold for the case of strongly q-fair languages generated by CDGS with non-constant communication. That is why the communication complexity of q-fair languages deserves to be studied separately. In what follows we focus on the time, space and communication complexity of these very particular class of languages. Below we give one of the most classical example, see [6], related to these types of languages.

Example 1 Let us consider the CDGS

$$\Gamma_1 = (\{S, A, A', B, B'\}, \{a, b\}, S, P_1, P_2, P_3, P_4)$$

with the components:

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

The communication graph is displayed in Figure 1.

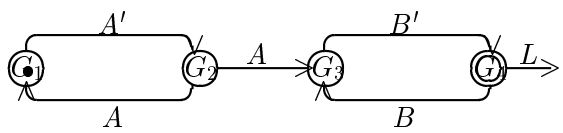

Figure 1: The communication graph associated to the CDGS, Γ_1.

The language generated by Γ_1, in the f mode of derivation, where $f \in \{t, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$ is: $L_f(\Gamma_1) = \{a^{2n}b^{2m} | n \geq 1, m \geq 1\} \in REG$. The *Szilard* language is $Sz(\Gamma_1) = \{(P_1P_2)^n(P_3P_4)^m | n \geq 1, m \geq 1\}$. If we impose the fairness conditions then we have:
$L_f(\Gamma_1, w-q) = L_f(\Gamma_1, s-q) = \{a^{2n}b^{2m} | n \geq 1, m \geq 1, |n-m| \leq q\} \notin REG$. Not being regular this language cannot be inferred from the communication graph

displayed in Figure 1. That is why we need another kind of machine to check the fairness conditions. Next we show that if the communication is not constant then the communication complexity of q-fair languages cannot be more than linear. Furthermore, fairness conditions can be checked in linear time and space by a multitape Turing machine, and in linear space and quadratic time by a one tape Turing machine. For the definitions of one tape or multitape Turing machine, the reader is referred to [16].

Theorem 3 $CD_rX(f, w-q) \cup CD_rX(f, s-q) \in COM(n)$.

Proof. In the case of grammar systems with regular and linear rules the Szilard and the communicational Szilard languages are equal. With respect to Theorem 1, for any word w that belongs to the weakly or strongly q-fair language, we have: $|\gamma_w(D)| = Com(w, \Gamma) = |\gamma_w^c(D)|$, where D is a minimal derivation of w. Hence, $sup\{|\gamma_w(D)|| \ w \in L_f(\Gamma, u-q), |w| = n\} = sup\{Com(w, \Gamma)| \ w \in L_f(\Gamma, u-q), |w| = n\} = Com_\Gamma(n) = sup\{|\gamma_w^c(D)||w \in L_f(\Gamma, u-q), |w| = n\} = O(n)$, where $u \in \{w, s\}$. □

Theorem 4 Let Γ be a CDGS$_r$X, for $X \in \{REG, LIN\}$. The weakly q-fair language generated by Γ, i.e., $L_f(\Gamma, w-q)$, can be accepted by a nondeterministic Turing machine with $r+1$ tapes in linear time and space. Moreover the next relation hold:

$$Space_T(n) \in O(Com_\Gamma), \quad Time_T \in O(Com_\Gamma).$$

Proof. Let $\Gamma = (N, T, S, P_1, \ldots, P_r)$ be a CDGS$_r$X, for $X \in \{REG, LIN\}$. Let $L_f(\Gamma, w-q)$ be the weakly q-fair language generated by Γ. Next we describe a nondeterministic $(r+1)$-tape Turing machine, with left, right, and stationary movements, that accepts $L_f(\Gamma, w-q)$. For the beginning the machine has on the first tape an input string w of length n, generated by the above CDGS$_r$X, for $X \in \{REG, LIN\}$, in f-mode of derivation, $f \in \{t\} \cup \{\leq k, = k, \geq k | k \geq 1\}$, followed by q symbols \$. Symbols from the right side of a production of the form $A \to xA'y$, or of the form $A \to A'y$, are marked in the input string by an index r, while the others are left unchanged. Each tape of the machine corresponds to one of the system components. At the start of the computation all heads are placed at the beginning of each tape. The machine starts nondeterministically with the component that contains the axiom. Let us assume that this is the component $i-1$. Therefore, the ith head of the machine writes the nonterminal symbol that appears on the right side of the starting production on the ith tape, letting the first head to read the symbol x from the input string, corresponding to a starting rule of the form $S \to xS'$, $S \to xS'x_r$, or to read no symbol in the case of starting rules of the form $S \to S'$, $S \to S'x_r$. From now on the machine simulates on the ith tape the derivation process done by the component $i-1$, in the f-mode, as follows. The first head reads the symbols x (or x_r) from the input tape, with respect to rules of the form $A \to xA'$, $A \to xA'x_r$ (or $A \to A'x_r$), and does not read any symbol with respect to rules of the form $A \to A'$, $A \to A'x_r$ of the component $i-1$. In the meantime, the ith head rewrites on the same cell the nonterminal reached by the rule, i.e. A'. All the other $(r-1)$ heads make no moves and write no symbols. When a communicational nonterminal has been reached, and

when the f-mode of derivation of the component $i-1$ ends, the machine nondeterministically jumps to the tape corresponding to the component to which the current component is communicating. Let us consider that this is the component $j-1$. In this moment the machine replaces the communicational nonterminal from the ith tape with the \$ symbol, in order to mark that the ith component has been applied once in the generative process. From now on the machine simulates the work of the component $j-1$ on the jth tape, in the same way as before. The process is repeated until the whole input string is read[2], i.e., the generative process performed by the CDGS$_r$X, for $X \in \{REG, LIN\}$ in the f-mode of derivation has been accomplished. In this moment the first head of the machine will be located on the symbol \$. On each ith tape there will be $\psi_D(i-1)$ symbols \$, that correspond to the number of contributions brought on the sentential form by the component $(i-1)$, $2 \leq i \leq r+1$. From now on the checking of the weak fairness condition, is performed as follows. All the heads, excepting the first one, simultaneously delete one by one the \$ symbols that have been written on each tape. When one of the heads gets at the beginning of the tape, i.e. the minimum number of the contributions of a component had been deleted, the first head starts to read the q symbols \$ from the first input tape. If these symbols have been read before the deletion of all \$ symbols from the other tapes it means that the weak q-fair restriction is not accomplished, therefore w will not be accepted by the machine. If the deletion of the \$ symbols from all tapes ends before or at the same time with the reading of the q symbols \$ from the first tape, w satisfies the weak q-fair restriction. Therefore the word will be accepted by the machine, as belonging to $L_f(\Gamma, w-q)$. The space needed by this machine to perform the computation cannot be more than the number of the communication steps performed by the grammar system, that is $Space_T(n) \in O(Com_\Gamma(n))$. The time required by the machine to accept and to check the weak q-fair condition is $O(Com_\Gamma(n))$, too. Due to Theorem 3, $Com_\Gamma \in O(n)$, so that $Space_T \in O(n)$ and $Time_T \in O(n)$. □

Theorem 5 Let Γ be a CDGS$_r$X, for $X \in \{REG, LIN\}$. The weakly *q-fair* language generated by Γ, i.e. $L_f(\Gamma, w-q)$, can be accepted by a nondeterministic Turing machine with one tape in linear space and quadratic time, i.e.

$$Space_T \in O(n), \quad Time_T \in O(n^2).$$

The above theorem is a direct consequence of Theorem 4 and of Savitch's theorem.

3.3 CDGS with Context-free Components

Theorem 6 $CD_*CF(f) = CF$, for $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$.

Corollary 7 $SZ(CDGR_*CF, f) = REG$, for $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$.

Corollary 8 $CF \subseteq COM(0)$.

Corollary 9 $CD_{r,c}CF(f, w-q) \in COM(0)$, where $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$, and c is the constant number of communication steps spent during the computation.

[2]Note that when all the communicational nonterminals have been spent, the symbols x_r left on the input tape, will be read by forbidding the other heads, excepting the first one, to move.

Corollary 10 The communication complexity of $L_f(\Gamma, w-q)$, $f \in \{*, =1, \geq 1\} \cup \{\leq k | k \geq 1\}$ generated by $CDGS_*CF$, for which the communication graph is a tree or a dag is 0.

Corollary 11 $CD_*LIN(f) \cup CD_*CF(f_1) \cup CD_{*,c}LIN/CF(f/f_1, w-q) \subset COM(0)$, $CD_{*,c}CF(f_2, w-q) \cup CD_{*,c}X(f, s-q) \subset COM(c)$, $X \in \{REG, LIN, CF\}$, for $f \in \{t, *\} \cup \{\leq k, = k, \geq k | k \geq 1\}$, $f_1 \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$ and $f_2 \in \{t\} \cup \{= k, \geq k | k \geq 1\}$.

In the case of non-constant communication we have.

Theorem 7 For each grammar system Γ with context-free components, and $w \in L_f(\Gamma)$, there exists a bijection $h : N \to N$ such that $|\gamma_w^c(D)| = h(|\gamma_w(D)|)$, where D is the the minimal derivation of w, $\gamma_w(D)$ and $\gamma_w^c(D)$ are the length of the derivation of the control word, γ_w, and of the communicational control word, γ_w^c, associated to w, respectively.

Proof. Let $\gamma_w^c = P_{i_1}^{n_1} P_{i_2}^{n_2} ... P_{i_k}^{n_k}$, be the communicational control word attached to w. In the case of CDGS with context-free components, γ_w^c is developed as a protocol tree attached to w. The number of sons at each level of this tree depends recursively on the number of sons of the previous levels, because communicational rules from different components are applied recursively, by using each time the same type of rules, that increases (linearly or exponentially) the number of communicational nonterminals used during the derivation. Consequently, at the end of the generative process the sum $n_1 + n_2 + ... + n_k$, will be a linear, polynomial or exponential function that depends on the length of the generated string. □

Next, we call the function h the *characterization function of the communicational Szilard language* $Szc(\Gamma, f)$.

Theorem 8 The class of languages generated by CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{= k, \geq k | k \geq 1\}$, for which the characterization function of the $Szc(\Gamma, f)$ language is linear, polynomial of rank p or exponential with the base p, has the communication complexity in $O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively.

Proof. With respect to Theorem 1, for any word $w \in L_f(\Gamma)$ of length n, we have
$$|\gamma_w(D)| = Com_\Gamma(|w|) = Com_\Gamma(n) \text{ and } |\gamma_w^c(D)| = O(|w|) = O(n).$$
With respect to Theorem 7, there exists a generative bijection $h : N \to N$, such that $|\gamma_w^c(D)| = h(|\gamma_w(D)|)$. Consequently, we have $O(n) = h(Com_\Gamma(n))$, so that $Com_\Gamma(n) = h^{-1}(O(n))$. Therefore, if h is a linear function then $Com_\Gamma(n) \in O(n)$, in the case that h is a polynomial function of rank p, then $Com_\Gamma(n) \in O(\sqrt[p]{n})$, while in the case that h is an exponential function of base p, then $Com_\Gamma(n) \in O(log_p n)$. □

Theorem 9 The class of languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{= k, \geq k | k \geq 1\}$, are generated by a nondeterministic Turing machine, with $r+1$ tapes, within $Space_T \in O(Com_\Gamma)$ and $Time_T \in \Theta(n)$.

Corollary 12 The class of languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{= k, \geq k | k \geq 1\}$ for which the characterization function of the $Szc(\Gamma, f)$ is linear, polynomial of rank p, or exponential with the base k, are recognizable by a nondeterministic Turing machine, with $r+1$ tapes, within $Space_T$ $O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively, and in $Time_T \in \Theta(n)$.

Furthermore, for the case of q-fair languages, the next theorem holds, [1].

Theorem 10 The class of q-fair languages generated by a CDGS$_r$CF in f-mode of derivation, $f \in \{t\} \cup \{= k, \geq k | k \geq 1\}$ for which the characterization function of $Szc(\Gamma, f)$ is linear, polynomial of rank p, or exponential with the base p, are recognizable by a $(k+1)$-tape nondeterministic Turing machine in $Space_T \in O(n)$, $O(\sqrt[p]{n})$, or $O(log_p n)$, respectively, and $Time_T \in \Theta(n)$.

4 Conclusion

The process of communication in the case of CDGS with regular and linear components is a lazy one. Furthermore, the communication can be lost, i.e., it has no efficiency in building communication complexity classes, in the case of CDGS with regular or linear components, without any fairness conditions. In the case of non-constant communication, for the case of weakly and strongly q-fair languages, the communication complexity cannot be more than linear. The communication can be lost too, for the case of languages generated by CDGS with context-free components, in the f mode of derivation, $f \in \{*, = 1, \geq 1\} \cup \{\leq k | k \geq 1\}$, or in the case of weakly q-fair languages generated by CDGS with constant communication. The communication is preserved in the case of CDGS with context-free rules, non-constant communication, weakly and strongly q-fair condition and in $f \in \{t\} \cup \{= k, \geq k | k \geq 1\}$ mode of derivation. Moreover, in this last situation we reached several complexity classes depending on the characterization function of the communicational Szilard language associated to a CDGS.

References

[1] L. Cojocaru. On the Time, Space, and Communication Complexity of q-fair Languages. In *Preproceedings of the 6th Descriptional Complexity of Formal Systems Workshop*, DCFS 2004, pp. 154-163, London, Ontario, Canada, July 26-28, 2004.

[2] E. Csuhaj-Varjú, J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics EIK* 26, 49-63, 1990.

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

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

[5] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems. *Handbook of Formal Languages*, Volume II, Chapter 4, edited by G. Rozenberg and A. Salomaa, Springer, Berlin, 155-213, 1997.

[6] J. Dassow, V. Mitrana. Fairness in Grammar Systems. Acta Cybernetica 12, 331-345, 1996.

[7] J. Dassow, V. Mitrana, Gh. Păun. Szilard Languages Associated to Cooperating Distributed Grammar Systems. *Studii si Cercetari Matematice*, 45, 403-413, 1993.

[8] M. Gheorghe, Gh. Păun. Further remarks on cooperating distributed grammar system. *Bull. Math. Soc. Sci. Math. Roumanie.* 34(82), 232- 245, 1990.

[9] J. Hromkovič, J. Kari, L. Kari, D. Pardubská, Two lower bounds on distributive generation of languages. *Fundamenta Informatica*, 25, 271-284, 1996.

[10] J. Hromkovič, J. Kari, L. Kari, Some hierarchies for the communication complexity measures of cooperating grammar systems. *Theoretical Computer Science*, 127, 123-147, 1994.

[11] D. Pardubská, On the power of communication structure for distributed generation of languages. *In Developments in Language Theory at the Crossroads of Mathematics, Computer Science and Biology*, Editors G. Rozenberg and A. Salomaa, Turku, Finland, 12-15, July, 1993.

[12] D. Pardubská, The Communication complexity hierarchy of parallel parallel communicating systems. IMYC'92, 1992.

[13] M. Lukàč. About two communication structures of PCGS. Master Thesis, Department of Computer Science, Faculty of Mathematics and Physics, Comenius University, Bratislava, 1992.

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

[15] R. Meersman, G. Rozenberg, D. Vermeir. Cooperating grammar systems. In: *Techn. Report 78-12*, Univ. Antwerp, Dept. Math., 1978.

[16] I. Sudkamp. Languages and Machines. An Introduction to the Theory of Computer Science. Addison-Wesley Ed., 1997.

[17] A. C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*, 209-213, 1979.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 126 - 140, 2004.

Parallel Communicating Pushdown Transducer Systems

Liliana Cojocaru, Carlos Martín-Vide

Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
liliana.cojocaru@estudiants.urv.es, cmv@astor.urv.es

Abstract

In this paper a new variant of translating devices is presented. We propose a new model based on the interconnection of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. We focus on the strategy of communication by stacks, i.e., the content of the pushdown memory of each component of the system is shared with each other component, and transferred into the pushdown memory of the component that asked for it. They communicate also by the output tapes, i.e., whenever a query symbol appears on the top of a stack, the string yielded so far, on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that inquired for it. We call these devices *Parallel Communicating Pushdown Transducer Systems* (henceforth PCPTS). Depending on the protocol of communication we define several variants of PCPTS, e.g. returning or not returning, centralized or not. The strategy of exchanging data through the pushdown memory increases the complexity of the outputted languages. We investigate the computational power of these systems by taking into consideration the computational power of parallel communicating pushdown automata systems. Descriptional complexity reasons suggest us to use PCPTS in DNA computing. Several examples illustrate the manner in which this application is performed.

1 Introduction

From the very beginning, due to their practical applicability, translating systems turned out to be a rich and interesting subject area in the field of computational linguistics. The main goal of this paper is to describe a new variant of translating devices obtained by improving parallel communicating pushdown automata systems with output capabilities. The mechanism could have interesting applications in computational morphology and phonology, in speech recognition, artificial intelligence or communication among agents, in splicing systems and in DNA computing.

Finite-state transducers have been introduced for the first time in [11]. Since then they have increasingly developed and diversified due to their flexibility in representing and generating a large size of structural data, by using time and space optimal memory, e.g. the transducer minimization algorithm [21]. They have been applied

especially in natural language processing, in fields such as computational morphology and phonology [14], lexical analyzers [15] or speech recognition [20]. They have been extended to algebraic transducers [16] and weighted finite-state transducers [22]. Watson-Crick transducers have been developed in order to manipulate DNA molecules. They are described in [23]. If a pushdown automaton is provided with output capability, the resulting machine is a pushdown transducer. It comes in the literature from [9], [10] and [12], with many other succeeding papers. Parallel communicating pushdown automata are systems composed of several pushdown automata working independently but communicating with each other by stacks, under a specified protocol of cooperation. The protocol of cooperation consists in exchanging strings that have been generated up to the request moment through the pushdown memory. They have been defined and investigated in [4], [5], [17], [18], [19] and [27] with forerunner related papers [1], [13]. If a parallel communicating pushdown automata system is improved with output capability the resulting device is called in this paper Parallel Communicating Pushdown Transducer Systems. The new model is based on the interconnection of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. The communication is done through stacks, i.e., the content of the pushdown memory of each component of the system is shared with each other component, and transferred into the pushdown memory of the component that asked for it. They communicate also by the output tapes, i.e., whenever a query symbol appears on the top of a stack the string yielded so far, on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that asked for it. Depending on the protocol of communication, we propose several variants of PCPTS, e.g. returning or not-returning, centralized or not. The study of the computational power and possible applications of such systems in DNA computing is another goal of this paper. The theoretical approaches are accomplished by several examples that illustrate the manner in which PCPTSs work.

2 Preliminaries

Our aim in this section is to introduce the main concepts related to pushdown transducers and parallel communicating transducer systems. A setting of the notations of concepts from the topic covered in the paper will be given, too. We assume the reader to be familiar with the basic notions of formal language theory. Let Σ^* be the set of words over an alphabet Σ, composed of a finite set of letters, let λ be the empty word, and let w^R and $|w|$, $w \in \Sigma^*$, be the reverse and the length of the word w, respectively. We denote by $\Sigma^+ = \Sigma^* - \{\lambda\}$ and by $|\Sigma|$, the cardinality of the set Σ. $\bar{\Sigma}$ is the twin alphabet of Σ, i.e. $\bar{\Sigma} = \{\bar{x} | x \in \Sigma\}$. A mapping $h : \Sigma^* \longrightarrow T^*$, defined by $h(\lambda) = \{\lambda\}$ and $h(x_1x_2) = h(x_1)h(x_2)$, for x_1, $x_2 \in \Sigma^*$, is called a morphism. If $T \subseteq \Sigma$, then a projection associated with T, is a morphism $h_T : \Sigma^* \longrightarrow T^*$, defined by $h_T(x) = x$, for all $x \in T$, and $h_T(y) = \lambda$, for all $y \in \Sigma - T$.

2.1 Pushdown Transducers

A pushdown transducer is obtained by permitting a pushdown automaton (henceforth PDA) to emit a string of output symbols on each transition.

Definition 1 A pushdown transducer (PDT) is an 8-tuple:

$$P = (Q,\ \Sigma,\ \Gamma,\ \Delta,\ \delta,\ q_0,\ Z_0,\ F)$$

where:

- Q is a set of states, Σ is the input alphabet, Γ is the alphabet of pushdown memory and Δ is the output alphabet (all these sets are finite),
- δ is a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ into the finite subsets of $Q \times \Gamma^* \times \Delta^*$,
- q_0 the initial state, $Z_0 \in \Gamma$ is the start symbol from the pushdown memory and $F \subseteq Q$, the set of final states .

A configuration of P is a 4-tuple (q, x, α, y), where q represents the current state of the finite control, x is the unread portion of the input, α is the content of the pushdown memory and y is the output string emitted up to this point. The leftmost symbol of α is the topmost pushdown symbol .

If $\delta(q, a, Z)$ contains (p, α, z), then we write:

(1) $$(q, ax, Z\gamma, y) \vdash (p, x, \alpha\gamma, yz), \qquad \text{for all } x \in \Sigma^*,\ y \in \Delta^*, \text{ and } \gamma \in \Gamma^*.$$

If $P = (Q,\ \Sigma,\ \Gamma,\ \Delta,\ \delta,\ q_0,\ Z_0,\ F)$ is a PDT, then the underlying pushdown automaton of P, is the pushdown automaton defined as:

$$A = (Q,\ \Sigma,\ \Gamma,\ \delta',\ q_0,\ Z_0,\ F),$$

where $(p, \alpha) \in \delta'(q, a, Z)$ if and only if $(p, \alpha, z) \in \delta(q, a, Z)$ for some $z \in \Delta^*$.

Using the relation (1), v is an output for $u \in L$, *by final states*, if:

(2) $$(q_0, u, Z_0, \lambda) \vdash^* (q, \lambda, \alpha, v), \quad \text{where } q \in F.$$

v is an output for $u \in L$, *by empty pushdown memory*, if:

(3) $$(q_0, u, Z_0, \lambda) \vdash^* (q, \lambda, \lambda, v), \quad \text{where } q \in Q.$$

Let P be a PDT and $L \subseteq \Sigma^*$ a language over Σ, recognized by the underlying pushdown automaton A. *The transduction realized by* P is a function

$$T_P : \Sigma^* \longrightarrow \Delta^* \quad \text{defined as:}$$

$$T_P = \{(u, v) |\ \text{such that } u \in \Sigma^* \text{ and } v \in \Delta^* \text{ satisfy one of the relations (2),(3)}\}.$$

Thus the transduction function associated to the transducer P returns, for each input word u, the set of all words generated in the output tape of P at the end of a successful computation on u.

A PDT is *deterministic* (DPDT for short), if the following conditions hold, for all $q \in Q$:

- $|\delta(q, a, Z)| \leq 1$ for each $a \in \Sigma \cup \{\lambda\}$;
- if $|\delta(q, \lambda, Z)| \neq 0$ then for all $a \in \Sigma$, $|\delta(q, a, Z)| = 0$.

2.2 Parallel Communicating Pushdown Transducer Systems

Definition 2 A parallel communicating pushdown transducer system of degree n ($pcpt(n)$ for short) is a system:

$$\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K),$$

where:

1. Σ is the input alphabet,
2. Δ is the output alphabet,

3. Γ is the alphabet of pushdown symbols.
4. For each $1 \leq i \leq n$, $T_i = (Q_i, \Sigma, \Gamma, \Delta, \delta_i, q_i, Z_i, F_i)$ is a pushdown transducer where Q_i is the set of states, $q_i \in Q_i$ is the initial state of the transducer T_i, Σ is the input alphabet, Γ is the pushdown alphabet, Δ is the finite output alphabet, $Z_i \in \Gamma$ is the initial contents of the pushdown memory, $F_i \subseteq Q_i$ is the set of final states of the transducer T_i, and δ_i is the transition mapping defined from $Q_i \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q_i \times \Gamma^* \times \Delta^*$.
5. $K \subseteq \{K_1, K_2, \ldots, K_n\} \subseteq \Gamma$ is the set of query symbols.

The pushdown transducers $T_1, T_2, ..., T_n$ are called the *components* of the system $\mathcal{T}$. If there exists only one $1 \leq i \leq n$, such that only the transducer T_i is allowed to query, then the system is called *centralized* (henceforth *cpcpt*(n)). The master of this system is the component T_i.

Definition 3 A PCPTS is said to be *deterministic* (DPCPTS for short), if the next conditions hold, for all $q \in Q_i$, $1 \leq i \leq n$:
- $|\delta_i(q, a, Z)| \leq 1$ for each $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$;
- if $|\delta_i(q, \lambda, Z)| \neq 0$ then for all $a \in \Sigma$, $Z \in \Gamma$, then $|\delta(q, a, Z)| = 0$.

A configuration of a PCPTS of degree n is a $4n$-tuple:

$$(s_1, x_1, \alpha_1, \gamma_1, s_2, x_2, \alpha_2, \gamma_2, \ldots, s_n, x_n, \alpha_n, \gamma_n)$$

where for all $1 \leq i \leq n$:
- $s_i \in Q_i$ is the current state of the component T_i,
- $x_i \in \Sigma^*$ is the remaining part of the input word that has not been read yet by T_i,
- $\alpha_i \in \Gamma^*$ is the contents of the i-th stack,
- $\gamma_i \in \Delta^*$ is the output string emitted up to that point by T_i.

The transition relations are defined on the set of all configurations of $\mathcal{T}$ as follows:
(4) $(s_1, x_1, B_1\alpha_1, \gamma_1, \ldots, s_n, x_n, B_n\alpha_n, \gamma_n) \vdash (p_1, y_1, \beta_1\alpha_1, \gamma_1^{'}, \ldots, p_n, y_n, \beta_n\alpha_n, \gamma_n^{'})$
where $B_i \in \Gamma, \alpha_i, \beta_i \in \Gamma^*$ and $\gamma_i, \gamma_i^{'} \in \Delta^*$ for all $1 \leq i \leq n$, iff one of the following two conditions holds, for all $1 \leq i \leq n$:
(4.i) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in \Sigma \cup \{\lambda\}, \gamma_i^{'} = \gamma_i z_i, (p_i, \beta_i, z_i) \in \delta_i(s_i, a_i, B_i)$,
(4.ii) for all $i, 1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K, \beta_i = B_{j_i}\alpha_{j_i}$, and $\gamma_i^{'} = \gamma_i\gamma_{j_i}$. For all other $r, 1 \leq r \leq n, \beta_r = B_r$, and $y_t = x_t$, $p_t = s_t$, for all $t, 1 \leq t \leq n$.
(5) $(s_1, x_1, B_1\alpha_1, \gamma_1, \ldots, s_n, x_n, B_n\alpha_n, \gamma_n) \vdash_r (p_1, y_1, \beta_1\alpha_1, \gamma_1', \ldots, p_n, y_n, \beta_n\alpha_n, \gamma_n')$
where $B_i \in \Gamma, \alpha_i, \beta_i \in \Gamma^*, \gamma_i, \gamma_i' \in \Delta^*$, $1 \leq i \leq n$, if one of the following two conditions holds, for all $1 \leq i \leq n$:
(5.i) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in \Sigma \cup \{\lambda\}, \gamma_i^{'} = \gamma_i z_i, (p_i, \beta_i, z_i) \in \delta_i(s_i, a_i, B_i)$,
(5.ii) for all $1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K, \beta_i = B_{j_i}\alpha_{j_i}, \beta_{j_i}\alpha_{j_i} = Z_{j_i}, \gamma_i^{'} = \gamma_i\gamma_{j_i}$, and $\gamma_{j_i} = \lambda$. For all the other $r, 1 \leq r \leq n, \beta_r = B_r$, and $y_t = x_t$, $p_t = s_t$, for all $t, 1 \leq t \leq n$.

A *pcpt*(n) system whose moves are all based on the relation $\vdash_r$ is said to be *a returning* (henceforth *rpcpt*(n)), and non-returning otherwise.

Note that, according to the definition of transitions sketched in (4) and (5), a PCPTS functions as follows: whenever a component has a query symbol on top of the pushdown memory, it gets into communication with the component to which the

query symbol belongs. The communication consists of replacing the query symbol with the entire content of the pushdown memory of the interrogated component, that is the sender. Simultaneously, the string yielded so far on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that queried for it. In the case of returning systems the pushdown memory of the component that sent the information returns to its initial symbol, while the output tape of the same component returns to the empty tape.
The reflexive and transitive closure of relation $\vdash$ ($\vdash_r$) is denoted by $\vdash^*$ ($\vdash_r^*$).
If the underlying pushdown automaton A_i, of the pushdown transducer T_i, is an extended one, for each $1 \leq i \leq n$, then the system $\mathcal{T}$, is said to be an extended PCPTS (henceforth EPCPTS). It means that the transition mappings δ_i are defined from finite subsets of $Q_i \times (\Sigma \cup \{\lambda\}) \times \Gamma^*$ to finite subsets of $Q_i \times \Gamma^* \times \Delta^*$, for all $1 \leq i \leq n$. Transitions for an EPCPT are defined in the same way as for PCPTS, with difference that extended systems may read words $B_i \in \Gamma^*$ instead of symbols. Note that PCPTS and EPCPTS are equivalent from a computational point of view.

Using the relation defined in (4), the set of words $v_i \in \Delta^*$, $1 \leq i \leq n$ is an output for $u \in \Sigma^*$, of a PCPTS with n components, if and only if:
(6) $(q_1, u, Z_1, \lambda, \ldots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \ldots, s_n, \lambda, \alpha_n, v_n), s_i \in F_i, 1 \leq i \leq n$ for a PCPTS accepting by final states, and :
(6') $(q_1, u, Z_1, \lambda, \ldots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \lambda, v_1, \ldots, s_n, \lambda, \lambda, v_n), s_i \in Q_i, 1 \leq i \leq n$ for a PCPTS accepting by empty pushdown memory .
Replacing the relation $\vdash^*$ with the relation $\vdash_r^*$ in (6) and in (6'), the definition of the output words, for a RPCPTS accepting by final states or by empty pushdown memory is obtained, respectively.

If $\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K)$ is $pcpt(n)$, then the underlying $pcpa(n)$ of $\mathcal{T}$ is defined as $\mathcal{A} = (\Sigma, \Gamma, A_1, A_2, \ldots, A_n, K)$, where A_i is the underlying PDA associated to T_i, for all $1 \leq i \leq n$. Let $\mathcal{T}$ be a PDT and $L \subseteq \Sigma^*$ a language over Σ, recognized by the underlying pushdown automaton $\mathcal{A}$.

Definition 4 Let $\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K)$, be a (nonreturning) PCPTS with n components. The **transduction function** of the ith component of the system $\mathcal{T}$ is defined as :

$$\mathrm{T}_i(u) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \cdots, s_i, \lambda, \alpha_i, v_i, \cdots, s_n, \lambda, \alpha_n, v_n), s_j \in F_j, 1 \leq j \leq n\},$$

for any $1 \leq i \leq n$, if the word u is accepted by final states, and

$$\mathrm{T}_i(u) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \lambda, v_1, \cdots, s_i, \lambda, \lambda, v_i, \cdots, s_n, \lambda, \lambda, v_n), s_j \in Q_j, 1 \leq j \leq n\},$$

for any $1 \leq i \leq n$, if the word u is accepted by empty pushdown memory.

If in the above definition the relation $\vdash_r$ is used instead of $\vdash$ then the transduction function of the ith component, for the returning case, denoted by TR_i is obtained. Thus each component T_i, $1 \leq i \leq n$, has associated a transduction mapping that returns for each input word u the set of all words yielded by that component.

The *language yielded by the component* T_i, $1 \leq i \leq n$ is defined as:

$$T_i(L) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \cdots, s_i, \lambda, \alpha_i, v_i, \cdots, s_n, \lambda, \alpha_n, v_n), s_j \in F_j, u \in L, 1 \leq j \leq n\},$$ or by

$T_i(L) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^*$
$(s_1, \lambda, \lambda, v_1, \cdots, s_i, \lambda, \lambda, v_i, \cdots, s_n, \lambda, \lambda, v_n), s_j \in Q_j, u \in L, 1 \leq j \leq n\}$.

If in the above definition the relation $\vdash_r$ is used instead of $\vdash$ then the definition of the language yielded by the ith component, for the returning case, denoted by $TR_i(L)$ is obtained. Due to the definition of PCPTS different components can output different languages. The *transduction system mapping* of a PCPTS, with n components, is defined as: $\mathrm{T} = (\mathrm{T}_1, \mathrm{T}_2, \cdots, \mathrm{T}_n)$, for non-returning case, and as $\mathrm{TR} = (\mathrm{TR}_1, \mathrm{TR}_2, \cdots, \mathrm{TR}_n)$, for the returning case.

3 On the Computational Power of PCPTS

The problem that arises now, is to study the type of the possible languages yielded by a PCPTS having as input an arbitrary language $L \subseteq \Sigma^*$. Let RE, CS, CF, LIN, and REG be the families of languages generated by arbitrary, context-sensitive, context-free, linear and regular grammars, respectively. The next inclusions hold:

(7) $$REG \subset LIN \subset CF \subset CS \subset RE$$

For each parallel communicating pushdown automata system we can build a PCPTS, having the same structure as the underlying system. That is why pushdown transducer systems will be able to accept and to generate at least the same type of languages. In [4] has been proved that the family of languages accepted by a PCPAS with two components equals the RE languages. Therefore, the family of languages yielded by PCPTS with only two components covers the the family of RE languages. Next we center upon the answers of the following questions:

"*Given an input language from a class X of the hierarchy (7), could it be possible to generate a language from a class Y such that $X \subset Y$?*" Furthermore,

"*Which is the smallest class of languages from the hierarchy (7) that could be used, as input language, by a PCPTS to cover the RE family ?*", and even more:

"*Which is the minimal number of components that a PCPTS should have in order to satisfy the above aims?*"

Answers to these questions will be given in the sequel. The following example is relevant for the computational power of PCPTS.

Example Let us consider a deterministic $pcpt(2)$, defined below:

$$\mathcal{T} = (\{a, c\}, \{Z_0, Z_1, a\}, T_1, T_2, \{K_1, K_2\}),$$

in which $T_i = (Q_i, \Sigma, \Gamma, \Delta, \delta_i, q_i, Z_i, F_i)$, $i \in \{1, 2\}$, are two pushdown automata, defined as: $T_1 = (\{q_1, s_1, r_1, p_1\}, \{a, c\}, \{Z_0, Z_1, a\}, \{a\}, \delta_1, q_1, Z_1, \{p_1\})$,
$T_2 = (\{q_2, s_2, p_2\}, \{a, c\}, \{Z_0, Z_1, a\}, \{a\}, \delta_2, q_2, Z_1, \{p_2\})$,

having the transition mappings :

1. $\delta_1(q_1, a, Z_1) = (s_1, Z_1, \lambda)$
2. $\delta_1(s_1, a, Z_1) = (s_1, aaZ_1, \lambda)$
3. $\delta_1(s_1, a, a) = (s_1, aaa, \lambda)$
4. $\delta_1(s_1, c, a) = (r_1, Z_0aaa, \lambda)$
5. $\delta_1(r_1, \lambda, a) = (r_1, \lambda, \lambda)$
6. $\delta_1(r_1, \lambda, Z_0) = (r_1, \lambda, \lambda)$
7. $\delta_1(r_1, \lambda, Z_1) = (p_1, K_2, \lambda)$
8. $\delta_1(p_1, \lambda, a) = (p_1, \lambda, a)$

1. $\delta_2(q_2, \lambda, Z_1) = (q_2, K_1, \lambda)$
2. $\delta_2(q_2, \lambda, a) = (q_2, K_1a, \lambda)$
3. $\delta_2(q_2, \lambda, Z_0) = (s_2, \lambda, \lambda)$
4. $\delta_2(s_2, a, a) = (s_2, \lambda, \lambda)$
5. $\delta_2(s_2, c, a) = (p_2, \lambda, \lambda)$
6. $\delta_2(p_2, \lambda, a) = (p_2, \lambda, \lambda)$
7. $\delta_2(p_2, \lambda, Z_1) = (r_2, \lambda, \lambda)$
8. $\delta_2(r_2, \lambda, a) = (r_2, \lambda, a)$

9. $\delta_1(p_1, \lambda, Z_1) = (p_1, \lambda, a)$ 9. $\delta_2(r_2, \lambda, Z_1) = (r_2, \lambda, a)$.

Taking as input the regular language: $L = \{a^n c \mid n \geq 1\}$ the above PCPTS will yield on both of the output tapes the following non-context-free language $L = \{a^{n^2-1} \mid n \geq 1\}$, by following the next transitions:

$$\underline{(q_1, a^n c, Z_1, \lambda, q_2, a^n c, Z_1, \lambda)} \vdash_{1,1} (s_1, a^{n-1}c, Z_1, \lambda, q_2, a^n c, K_1, \lambda)$$
$$\vdash_{2,1} (s_1, a^{n-2}c, aaZ_1, \lambda, q_2, a^n c, K_1, \lambda)$$
$$\vdash_{3,2}^{*(n-2)} (s_1, c, a^{2(n-1)}Z_1, \lambda, q_2, a^n c, a^{2(n-1)}Z_1 ... a^2 Z_1, \lambda)$$
$$\vdash_{4,2} (r_1, \lambda, Z_0 a^{2n} Z_1, \lambda, q_2, a^n c, K_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{6,3} (r_1, \lambda, a^{2n} Z_1, \lambda, s_2, a^n c, a^{2n} Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,4}^{*n} (r_1, \lambda, a^n Z_1, \lambda, s_2, c, a^n Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,5} (r_1, \lambda, a^{(n-1)} Z_1, \lambda, p_2, \lambda, a^{(n-1)} Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,6}^{*(n-1)} (r_1, \lambda, Z_1, \lambda, p_2, \lambda, Z_1 a^{2(n-1)} .. a^2 Z_1, \lambda)$$
$$\vdash_{7,7} (p_1, \lambda, K_2, \lambda, r_2, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda) \vdash$$
$$\vdash (p_1, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda, r_2, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda)$$
$$\vdash_{8,8}^{*2(n-1)} (p_1, \lambda, Z_1 ... a^2 Z_1, a^{2(n-1)}, r_2, \lambda, Z_1 a ... a^2 Z_1, a^{2(n-1)})$$
$$\vdash_{9,9} (p_1, \lambda, a^{2(n-2)} Z_1 .. a^2 Z_1, a^{2(n-1)} a, r_2, \lambda, a^{2(n-2)} Z_1 .. a^2 Z_1, a^{2(n-1)} a)$$
$$\vdash_{8,8}^{*2(n-2)} (p_1, \lambda, Z_1 a^{2(n-3)}_{\cdots} a^2 Z_1, a^{2(n-1)} a a^{2(n-2)}, r_2, \lambda, Z_1 a^{2(n-3)}_{\cdots} a^2 Z_1, a^{2(n-1)} a a^{2(n-2)})$$
$$\vdash_{9,9} \ldots \vdash_{8,8}^{*2(n-3)} \ldots \vdash_{9,9} \ldots \vdash_{8,8}^{*2(n-4)} \ldots \vdash_{9,9} \ldots \vdash_{8,8}^{*4} \ldots \vdash_{9,9} \ldots \vdash_{8,8}^{*2} \vdash_{9,9}$$
$$\underline{(p_1, \lambda, \lambda, a^{(n-1)} a^{2(n-1)} a^{2(n-2)} \ldots a^4 a^2, r_2, \lambda, \lambda, a^{(n-1)} a^{2(n-1)} a^{2(n-2)} \ldots a^4 a^2)}$$

The output is: $a^{(n-1)} a^{2(n-1)} a^{2(n-2)} \ldots a^4 a^2 = a^{2(n-1)+\ldots+4+2+n-1} = a^{n^2-1}$.

In order to express the computational power of PCPTS we will introduce several notations and basic results related to the *twin shuffle* operation on strings.

Definition 5 1) For two strings $x, y \in \Sigma^*$ and two symbols $a, b \in \Sigma$, the **shuffle** operation is defined as: (i) $x \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup x = \{x\}$,

(ii) $ax \sqcup\!\sqcup by = a(x \sqcup\!\sqcup by) \cup b(ax \sqcup\!\sqcup y)$.

2) The **shuffle of two languages** $L_1, L_2 \in \Sigma^*$ is:

$$L_1 \sqcup\!\sqcup L_2 = \bigcup_{x \in L_1, y \in L_2} x \sqcup\!\sqcup y.$$

3) The **twin shuffle language** over Σ, is defined as:

$$\mathrm{TS}_\Sigma = \bigcup_{x \in \Sigma^*} x \sqcup\!\sqcup \bar{x}.$$

The following representation of the family RE is well known from [23]:

Theorem 1.
Each recursively enumerable language $L \subseteq T^*$ can be written as $L = h_T(TS_\Sigma \cap R)$, where Σ is an alphabet, R is a regular language, and h_T is a projection associated with $T \subseteq \Sigma$.

Related to the computational power of PCPTS, the next result from [4] is useful in our considerations:

Theorem 2.
1. A language $L \subseteq T^*$ is recursively enumerable if and only if $L = h_T(Rec_r(\mathcal{A}))$, where $\mathcal{A}$ is a parallel communicating pushdown automata system[1] of type $x(2)$ with

[1]For the definition and more information related to these systems the reader is referred to [4].

$x \in \{rcpcpa, rpcpa\}$.
2. A language $L \subseteq T^*$ is recursively enumerable iff $L = h_T(Rec(\mathcal{A}))$, where $\mathcal{A}$ is a $cpcpa(2)$.

Due to the above theorems we have:

Theorem 3.
For each recursively enumerable language L there exists language $R \in REG$ and an automata system $\mathcal{T}$ of type $x(2)$, with $x \in \{rcpcpt, rpcpt, cpcpt\}$ such that $\mathrm{T}_{\mathcal{T}}(R) = (L, \lambda)$.

Proof. In the proof of Theorem 2 it has been shown that there exists a PCPAS of type $x(2)$, with $x \in \{rcpcpa, rpcpa, cpcpa\}$, that accepts the language $TS_\Sigma \cap R$, where $R \in REG$. With respects to Theorem 1, for each language $L \in RE$, there exists a projection h_T, $T \subseteq \Sigma$, and a language $R \in REG$, such that the image of $TS_\Sigma \cap R$ through h_T is the language $L \in RE$. With respect to Theorem 1 and Theorem 2 for each language $L \in RE$ there exists a language $R \in REG$, such that we can build a PCPTS, $\mathcal{T}$ of type $x(2)$, with $x \in \{rcpcpt, rpcpt, cpcpt\}$, that for each word $w \in R$, if $w \in TS_\Sigma$, $\mathcal{T}$ yields on its first output tape the word $h_T(w)$ (by simulating the projection h_T), and issues the empty word otherwise. The other component will output only the empty word. □

4 How to Apply PCPTS - A Descriptional Complexity Reasoning

As we have seen in the last section PCPTS are able to output RE languages having as input very simple REG languages. This result allows PCPTS to be used to simulate natural phenomena in which very simple data, seen as strings of REG languages, is processed into more complicated CS languages. Gene assembly in ciliates can be considered such a string processing phenomenon.

4.1 The Gene Assembly Process in Ciliates

Ciliates are single cell organisms that have two types of nuclei: *micronuclei* and *macronuclei*. The intramolecular process of transformation of a micronucleus into a macronucleus is known in literature as *gene assembly process in ciliates*. Micronuclear genes are composed of combinations of residual segments, called *Internal Eliminated Segments* (henceforth IES) and active segments, called *Macronuclear Destinated Segments* (henceforth MDS). The prominent feature of the gene assembly process consists in the very spectacular manner in which, during the transformation of micronuclear genes into macronuclear genes, the MDS regions are spliced and the IES portions are excised. The splicing process is done in some weaker points of the micronuclear genes, named *pointers*. These pointers are sequences of nucleotides, that bound the MDS components. The MDS breaking and reassembling is done such that at the end of the gene assembly process the yielded macronuclear genes will contain no residual segment. This procedure is performed with respect to three

molecular patterns that appear in the structure of a micronuclear gene. Each time a direct repeat pattern of pointers (p, p), i.e., no other pointer parts the two p pointers, appears in the structure of a micronuclear gene then the *ld*-(loop, direct repeat)-excision operation is performed. If an inverted repeat pattern of the form $(p, \bar{p})$ is found then the *hi*-(hairpin, inverted repeat)-excision/reinsertion operation is performed. In the case that the micronuclear gene contains an alternating direct repeat pattern of pointers (p, r), i.e., an overlapping structures of the form (p, p) and (r, r), then the *dlad*-(double loop, alternating direct repeat)-excision/reinsertion operation is carried on. From a computational point of view the three molecular operations *ld, hi*, and *dlad* have been studied in several papers, e.g. [8], [7], [24], [26], [6]. Our model has been inspired from the theoretical interpretation of the gene assembly process given in [7], and is briefly described below.

Let Π_k be the *pointers alphabet*, formed by the micronuclear gene pointers, usually denoted by $\Pi_k = \{2, 3, ..., k, \bar{2}, \bar{3}, ..., \bar{k}\}$. A string $\pi \in \Pi_k^*$ is called *legal* if each symbol p from π, is duplicated either by p, or by $\bar{p}$, and at most two occurrences of p (or $\bar{p}$) are allowed in π. Three operations can be defined on the sets of all legal strings over the alphabet Π_k:

1. If $\pi = \pi_1 pp\pi_2$ then $\mathbf{ld}_p$ (the *string negative rule* for p) is defined as: $\mathbf{ld}_p(\pi) = \pi_1\pi_2$.
2. If $\pi = \pi_1 p\pi_2\bar{p}\pi_3$ then $\mathbf{hi}_p$ (the *string positive rule* for p) is defined as: $\mathbf{hi}_p(\pi) = \pi_1\mathbf{rs}(\pi_2)\pi_3$, where $\mathbf{rs}(\pi_2)$ is the *reversed switch*[2] of π_2.
3. If $\pi = \pi_1 p\pi_2 r\pi_3 p\pi_4 r\pi_5$ then $\mathbf{dlad}_{p,r}$ (the *string double rule* for p and r) is defined as: $\mathbf{dlad}_{p,r}(\pi) = \pi_1\pi_4\pi_3\pi_2\pi_5$.

A *string pointer reduction system over* Π_k, denoted by $SPRS_{\Pi_k}$, is a set $SPRS_{\Pi_k} = \{\mathbf{ld}_p, \mathbf{hi}_p, \mathbf{dlad}_{p,r} | p, r \in \Pi_k\}$. We denote by $D = (\pi; \rho_1, \rho_2, ..., \rho_l)$, $l \geq 1$, $\rho_i \in SPRS_{\Pi_k}$, $\pi \in \Pi_k^*$, π a legal string, the *reduction of* π by applying the rules ρ_i, $0 \leq i \leq l$, in the order of their appearance in the scheme. We say that D is a *successful* reduction scheme for a legal string π, if the resulting string after the scheme application on π is the empty string. For more information related to properties of *reduction schemes*, the reader is referred to [7]. The string pointer reduction system described above is a formalization of the splicing phenomena that take place during the gene assembly in ciliates. In [7] it has been shown that the $\mathbf{ld}_p$ operation simulates the *ld* operation, $\mathbf{hi}_p$ simulates the *hi* operation, and finally $\mathbf{dlad}_{p,r}$ simulates the *dlad* operation. The successful scheme reduction of a legal string is convertible into a successful strategy for a realistic MDS descriptor (that is the theoretical representation of the IES/MDS structure of a micronuclear gene). A successful strategy for a realistic MDS descriptor represents a gene assembly procedure that has as result a successful transformation of a micronuclear gene into a macronuclear one, i.e., no pointer will appear, in the structure of the resulted macronuclear gene at the end of the molecular computation. In [2] we described a computational model based on Parallel Communicating Finite Transducer Systems (henceforth PCFTS[3]) that performs the above molecular operations. In the next section, descriptional reasonings suggest us that PCPTS can be used better than PCFTS to simulate the gene assembly process in ciliates.

[2]For instance $\mathbf{rs}(\bar{3}4\bar{2}234) = \bar{4}\bar{3}\bar{2}2\bar{4}3$.

[3]For the definition of PCFTS the reader is referred to [3].

4.2 PCPTS - A Descriptional Complexity Analysis

In the following we propose two $pcpt(2)$ that simulate **hi** and **dlad** operations. Because finite transducers/pushdown transducers can check the "legality" of a given string, next we will consider these operations defined only on legal strings.

Simulation 1 The $\mathbf{hi}_p$ operation
Let $T = (\Pi_k, \Pi_k, \{Z_0\}, T_1, T_2, K)$ be a *cpcpt*, with the components
$T_1 = (\{q_0, q_p, q_{\bar{p}}, q_{[p]}, q_{[\bar{p}]}, q_f\}, \Pi_k, \Pi_k, \Pi_k \cup \{Z_0, Z_1\}, \delta_1, q_0, Z_0, \{q_0, q_f\})$
$T_2 = (\{q_0, s_p, s_{\bar{p}}\}, \Pi_k, \Pi_k, \{Z_1\}, \delta_2, q_0, Z_0, \{q_0, s_p, s_{\bar{p}}\})$
and δ mappings are defined, for all $x \in \Pi_k$, as follows:

1. $\delta_1(q_0, x, Z_0) = \{(q_0, Z_0, x)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
1'. $\delta_2(q_0, x, Z_1) = \{(q_0, Z_1, \lambda)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
2. $\delta_1(q_0, p, Z_0) = \{(q_p, Z_0 Z_1, \lambda)\}$,
2'. $\delta_2(q_0, p, Z_1) = \{(s_p, Z_1, p)\}$,
3. $\delta_1(q_0, \bar{p}, Z_0) = \{(q_{\bar{p}}, Z_0 Z_1, \lambda)\}$,
3'. $\delta_2(q_0, \bar{p}, Z_1) = \{(s_{\bar{p}}, Z_1, \bar{p})\}$,
4. $\delta_1(q_p, x, Z_0) = \{(q_p, Z_0 x, \lambda)\}$, $x \in \Pi_k - \{\bar{p}\}$;
4'. $\delta_2(s_p, x, Z_1) = \{(s_p, Z_1, x)\}$, $x \in \Pi_k$;
5. $\delta_1(q_p, \lambda, Z_0) = \{(q_f, K_2, \lambda)\}$;
5'. $\delta_2(s_{\bar{p}}, x, Z_1) = \{(s_{\bar{p}}, Z_1, x)\}$, $x \in \Pi_k$;
6. $\delta_1(q_{\bar{p}}, x, Z_0) = \{(q_{\bar{p}}, Z_0 x, \lambda)\}$, $x \in \Pi_k - \{p\}$;
6'. $\delta_2(s_p, \lambda, Z_1) = \{(s_p, Z_1, \lambda)\}$;
7. $\delta_1(q_{\bar{p}}, \lambda, Z_0) = \{(q_f, K_2, \lambda)\}$;
7'. $\delta_2(s_{\bar{p}}, \lambda, Z_1) = \{(s_{\bar{p}}, Z_1, \lambda)\}$.
8. $\delta_1(q_p, \bar{p}, Z_0) = \{(q_{[\bar{p}]}, \lambda, \lambda)\}$;
9. $\delta_1(q_{[\bar{p}]}, \lambda, x) = \{(q_{[\bar{p}]}, \lambda, \bar{x})\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
10. $\delta_1(q_{[\bar{p}]}, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$;
11. $\delta_1(q_{\bar{p}}, p, Z_0) = \{(q_{[p]}, \lambda, \lambda)\}$;
12. $\delta_1(q_{[p]}, \lambda, x) = \{(q_{[p]}, \lambda, \bar{x})\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
13. $\delta_1(q_{[p]}, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$;
14. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;

The above system works as follows: for the beginning both components read symbols from the input tape until they reach p or $\bar{p}$. Only the first component outputs the symbols that have been read. If the pointers p and $\bar{p}$ do not exist in the input string, then the first component yields on its output tape the (whole) input string, while the second component yields the empty string. In this case, the system ends the computation in the pair of states: (q_0, q_0). When both components read the first occurrence of the pointer $p/\bar{p}$, the first component reaches the state $q_p/q_{\bar{p}}$, and the second one reaches the state $s_p/s_{\bar{p}}$. From now on, each symbol that will be read by the first component in the state $q_p/q_{\bar{p}}$, will be memorized in the pushdown memory, without any output. Each symbol that will be read by the second component (including $p/\bar{p}$) will be yielded on the second output tape. If the twin symbol of p (i.e. $\bar{p}$) or the twin symbol of $\bar{p}$ (i.e. p) is not found in the input string then the components communicate with each other in order to let the first component to

output the (whole) input on the first tape. Note that, in this case the first component cannot query during the computational process, otherwise the query symbol will be replaced by the initial symbol of the stack of the second component, fact that will block the computation, (the first mapping δ_1 not being defined in the state $q_p/q_{\bar{p}}$ for this symbol). In this situation the computation ends in the pair of states (q_f, s_p), if p is found, and in the pair of states $(q_f, s_{\bar{p}})$, if $\bar{p}$ is found. When the pair $(p, \bar{p})/(\bar{p}, p)$ is found the system changes the pair of states $(q_p, s_p)/(q_{\bar{p}}, s_{\bar{p}})$ into the pair $(q_{[\bar{p}]}, s_p)/(q_{[p]}, s_{\bar{p}})$. From now on the first component yields on its output tape the reversed switch of π_2 (that has been stored in its pushdown memory) reading no input symbol. When the first component reaches the bottom of the stack, it begins to read and to output the rest of the symbols from the input tape. During all this time the second component reads all the input symbols, having no output. The system ends the computation in the pair of states (q_f, s_p), if p is found before $\bar{p}$, and in the pair $(q_f, s_{\bar{p}})$, if $\bar{p}$ is found before p. Furthermore, the above system makes a simultaneous searching for both of the pairs $(p, \bar{p})$, and $(\bar{p}, p)$, while the $rpcft(2)$ presented in [2] performs this operation only for the pair $(p, \bar{p})$. We have to eliminate the rules 3, 6, 7, 11, 12, 13, 3', 5', 7', and the states $q_{\bar{p}}$, $q_{[p]}$, $s_{\bar{p}}$, in order to obtain a *cpcpt*(2) that performs the same task as the $rpcft(2)$ presented in [2]. Therefore, in what follows we compare the descriptional complexity of the $rpcft(2)$ with the reduced model of the *cpcpt*(2) presented above.

In the **hi** simulation displayed in [2], the $rpcft(2)$ that performs this operation needs 8 states for each component, $5|\Pi_k| + 3$ rules for the first component, and $4(|\Pi_k|+1)$ rules for the second one. The reduced model of the *cpcpt*(2) presented here needs only 4 states for the first component, 2 states for the second one, $4|\Pi_k|-3$ rules for the first component, and $2|\Pi_k|$ rules for the second one. The $rpcft(2)$ needs $|\pi_2|+2$ communication steps, while the above *cpcpt*(2) needs only one communication, used only in the situation when the twin symbol of p (or $\bar{p}$) is not found in the input string.

Simulation 2 The $\mathbf{dlad}_{p,r}$ operation

The next *rpcpt* system performs the **dlad** operation:

$T = (\Pi_k, \Pi_k, \{Z_0\}, T_1, T_2, K)$ be a *cpcpt*, with the components

$T_1 = (\{q_r, q_f, q_0, q_1, q_2, q_3, q_4, q_5\}, \Pi_k, \Pi_k \cup \{Z, Z_0, Z_1, Z_2, \bar{Z}, \bar{\bar{Z}}, \hat{Z}\}, \delta_1, q_0, Z_1, \{q_0, q_f\})$

$T_2 = (\{s_0, s_p, s_f, s_r, s_{[r]}, s_{[p]}, s_{[\bar{r}]}\}, \Pi_k, \Pi_k, \{Z, Z_2, \bar{\bar{Z}}\}, \delta_2, s_0, Z_2, \{s_0, s_f, s_{\bar{r}}\})$

and δ mappings defined, for all $x \in \Pi_k$, as follows:

1. $\delta_1(q_0, \lambda, Z_1) = \{(q_1, Z_0Z_1, \lambda)\}$, 1'. $\delta_2(s_0, x, Z_2) = \{(s_0, Z_2, \lambda)\}$, $x \neq p$;
2. $\delta_1(q_1, \lambda, Z_0) = \{(q_1, Z_0, \lambda)\}$, 2'. $\delta_2(s_0, \lambda, Z_2) = \{(s_f, K_1, \lambda)\}$,
3. $\delta_1(q_1, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$, 3'. $\delta_2(s_0, p, Z_2) = \{(s_p, Z_2, \lambda)\}$,
4. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, 4'. $\delta_2(s_p, x, Z_2) = \{(s_p, Z_2, \lambda)\}$,$x \notin \{p, \bar{p}\}$
5. $\delta_1(q_1, \lambda, Z_0Z_1) = \{(q_r, K_2, \lambda)\}$, 5'. $\delta_2(s_p, \{p, \bar{p}\}, Z_2) = \{(s_f, K_1, \lambda)\}$,
6. $\delta_1(q_r, \lambda, Z) = \{(q_r, Z, \lambda)\}$, 6'. $\delta_2(s_f, x, Z_0) = \{(s_f, Z_2, \lambda)\}$, $x \neq p$;
7. $\delta_1(q_r, \lambda, Z_1) = \{(q_r, K_2, \lambda)\}$, 7'. $\delta_2(s_f, x, Z_2) = \{(s_f, Z_2, \lambda)\}$, $x \neq p$;
8. $\delta_1(q_r, \lambda, \bar{Z}) = \{(q_f, Z_1, \lambda)\}$, 8'. $\delta_2(s_f, x, \bar{\bar{Z}}) = \{(s_f, \bar{\bar{Z}}, \lambda)\}$;
9. $\delta_1(q_r, \lambda, \bar{\bar{Z}}) = \{(q_1, \bar{\bar{Z}}, \lambda)\}$, 9'. $\delta_2(s_f, \lambda, Z_2) = \{(s_f, Z_2, \lambda)\}$,
10. $\delta_1(q_1, \lambda, \bar{\bar{Z}}) = \{(q_1, \bar{\bar{Z}}, \lambda)\}$, 10'. $\delta_2(s_p, r, Z_2) = \{(s_r, Z_2, \lambda)\}$,
11. $\delta_1(q_1, x, \bar{\bar{Z}}) = \{(q_1, \hat{Z}, x)\}$, 11'. $\delta_2(s_r, x, Z_2) = \{(s_{[r]}, Zx, \lambda)\}$,

12. $\delta_1(q_1, x, \hat{Z}) = \{(q_1, \hat{Z}, x)\}$, 12'. $\delta_2(s_{[r]}, x, Z_2) = \{(s_{[r]}, K_1 x, \lambda)\}$,
13. $\delta_1(q_1, p, \hat{Z}) = \{(q_2, K_2, \lambda)\}$, 13'. $\delta_2(s_{[r]}, \lambda, Z) = \{(s_{[r]}, Z, \lambda)\}$,
14. $\delta_1(q_2, \lambda, Z_2) = \{(q_3, \lambda, \lambda)\}$, 14'. $\delta_2(s_{[r]}, \{\bar{p}, r\}, Z) = \{(s_f, \bar{Z}, \lambda)\}$,
15. $\delta_1(q_3, \lambda, x) = \{(q_3, \lambda, x)\}$, 15'. $\delta_2(s_{[r]}, p, Z) = \{(s_{[p]}, \bar{\bar{Z}}, \lambda)\}$,
16. $\delta_1(q_3, x, Z_1) = \{(q_3, Z_1, x)\}$, $x \neq r$, 16'. $\delta_2(s_{[p]}, x, Z_2) = \{(s_{[p]}, Z_2, x)\}$,
17. $\delta_1(q_3, r, Z_1) = \{(q_4, \lambda, \lambda)\}$, 17'. $\delta_2(s_{[p]}, \bar{r}, Z_2) = \{(s_f, K_1, \lambda)\}$,
18. $\delta_1(q_4, x, Z_1) = \{(q_4, Z_1, \lambda)\}$, $x \neq r$, 18'. $\delta_2(s_{[p]}, r, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
19. $\delta_1(q_4, r, Z_1) = \{(q_5, Z_1, \lambda)\}$, 19'. $\delta_2(s_{[\bar{r}]}, x, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
20. $\delta_1(q_5, x, Z_1) = \{(q_f, Z_1, x)\}$, 20'. $\delta_2(s_{[\bar{r}]}, \lambda, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
21. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, 21'. $\delta_2(s_{[\bar{r}]}, \lambda, Z_2) = \{(s_{[\bar{r}]}, K_1 Z_2, \lambda)\}$.

Briefly, the above system works in two stages. In the first stage the second transducer makes a search, over the whole input string, for the alternating direct repeat pattern of pointers (p, r). During all this time the first transducer reads no symbol, and has no output. If this pattern is not found the first transducer outputs the whole input string on the first output tape. When this pattern is found the second transducer yields the result of the $\mathbf{dlad}_{p,r}$ operation. This will be done in the second stage of the simulation.

In more details, the system works as follows: for the beginning only the second component reads symbols from the input tape until the pointer p is found, without any output. If this pointer is not found, then the (whole) input string will be yielded on the output tape of the first component (due to rules 2', 3, and 4). When the symbol p is reached the current state of the second component is changed into the state s_p (rule 3'). From now on the second component continues the searching of r. If the pointers p or $\bar{p}$ are found before r, then the $\mathbf{dlad}_{p,r}$ operation cannot be performed. Thus the (whole) input string will be outputted by the first pushdown transducer (due to rules 5', 3, and 4). If r is found (before of p or $\bar{p}$), then the state s_p is changed into s_r. Afterward, the string placed between the pointers r and p, i.e., π_3 (see Section 4.1), is memorized in the pushdown memory of the second transducer. This process is performed as follows: in the state s_r the second transducer memories the first symbol from π_3, and changes the state s_r into the state $s_{[r]}$ (due to rule 11'). In this moment the first pushdown transducer is obliged to query (rule 5), the second pushdown transducer not being defined in the state $s_{[r]}$ with the top of pushdown memory set on the pushdown symbol Z. Due to the query process, the content of the pushdown memory of the second transducer will be discharged into the pushdown memory of the first transducer, and the pushdown memory of the second transducer is reduced to Z_2, due to the returning character of the system. In the state $s_{[r]}$, having on the top of pushdown memory the symbol Z_2, the second transducer is able now to memorize the next symbol from π_3 and to ask for the content of the first pushdown memory (rule 12'), in order to build in its memory the image of the string π_3, and not the mirror image of it, as it usually happens due to the structure of the pushdown memory. After the query step is performed, the pushdown memory of the first transducer returns to Z_1, due to the returning character of the system. In the next step the first transducer asks for the content of the pushdown memory of the second transducer (rule 7), in order to give freedom to the second one to memories the next symbol from π_3 and to make the concatenation

between the symbol that has been currently read and the substring of π_3 that had been already read and memorized into the pushdown memory of the first transducer (rule 12'). This process continues until the whole substring π_3 has been read and memorized into the pushdown memory of the second transducer. Afterwards, if in the search procedure of the second transducer the pointer $\bar{p}/r$ is found before the second occurrence of p the $\mathbf{dlad}_{p,r}$ operation cannot be performed. So that the (whole) input string will be outputted by the first pushdown transducer (due to rules 14', 8, 7, and 4). When the second occurrence of p is found, the state $s_{[r]}$ is changed into $s_{[p]}$ and the second transducer starts to yield the image of a possible π_4 (see Section 4.1). If the pointer $\bar{r}$ is found instead of r, then the second transducer queries the first one in order to let it output the whole input string. If the second occurrence of r is found, then an alternating direct repeat pattern of pointers (p, r) exists in the string, so that the result of the $\mathbf{dlad}_{p,r}$ operation will be yielded on the first output tape, according to the rules from 11 to 21.

If we compare the above $rpcpt(2)$ system with the $rpcft(2)$ presented in [2], from a descriptional complexity point of view, we will observe that the former one is more efficient than the latter one. Thus $rpcft(2)$, presented in [2], needs $|\Pi_k| + 7$ states, and $8|\Pi_k| + A_k^2 + 5$ rules for the first component, 9 states and $6|\Pi_k| + 9$ rules for the second component. The number of communications is fixed to 3. The above $rpcpt(2)$ needs 8 states and $8|\Pi_k| + 11$ rules for the first component, 7 states and $9(|\Pi_k| + 1)$ rules for the second one, and $|\pi_3| + 2$ communication steps. Furthermore, the system makes a complete searching for the pointers p and r, and it yields the output for the all situations.

All the observations done along the systems described in this section, argue that PCPTS turns out to be very efficient in implementing complex molecular operations, from a descriptional point of view.

5 Conclusions

In this paper we have introduced a new translating device called Parallel Communicating Pushdown Transducer Systems. They are systems composed of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. We focus on the strategy of communication by stacks, and by the output tapes. The protocol of collaboration is controlled by query symbols. They specify which component has to send the content of the pushdown memory and the content of the output tape, and which component has to receive them. The strategy of data exchanging through the pushdown memory and through the output tape allows each component to output complex information. Therefore the complexity of the yielded languages is substantially increased. The computational power of the new device has been investigated by taking into consideration the computational power of parallel communicating pushdown automata systems. A comparison between them and parallel communicating finite transducer systems has been done, too. Even if they are able to yield at least the same classes of languages as parallel communicating finite transducer systems do, they are more efficient from a descriptional complexity point of view.

References

[1] A. O. Buda. Multiprocessor Automata, *Information Processing Letters*, 25:257-261, 1977.

[2] L. Cojocaru. Simulating the Process of Gene Assembly in Ciliates. In *Proceedings of the* 9th *International Conference on Implementation and Application of Automata*, Queen's University, Kingston, Ontario, Canada, July 22-24, 2004.

[3] E. Csuhaj-Varjú, C. Martín-Vide, and V. Mitrana. Parallel Communicating Finite Transducer Systems. In *Language and Computers: Studies in Practical Linguistics*, 47, 9-23, 2002.

[4] E. Csuhaj-Varjú, C. Martin-Vide, V. Mitrana, and G. Vaszil. Parallel Communicating Pushdown Automata Systems. *International Journal of Foundations of Computer Science* 11, No. 4, 633-650, 2000.

[5] J. Dassow and V. Mitrana. Stack Cooperation in Multi-Stack Pushdown Automata, *Journal of Computer and System Sciences* 58, 611-621, 1999.

[6] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, and G. Rozenberg. *Computation in Living Cells, Gene Assembly in Ciliates.* Springer-Verlag, 2004.

[7] A. Ehrenfeucht, I. Petre, D.M. Prescott, and G. Rozenberg. Universal and Simple Operations for Gene Assembly in Ciliates. In *Where Mathematics, Computer Science, Linguistics and Biology Meet*, C.Martin-Vide, V. Mitrana (eds.), 329-342, Kluwer Academic Publishers, Dordrecht/Boston, 2001.

[8] A. Ehrenfeucht, D.M. Prescott, and G. Rozenberg. Computational Aspects of Gene (Un)scrambling in Ciliates. In *Evolution as Computation*, L. Landweber, E. Winfree (eds.), 45-86, Springer Verlag, Berlin, Heidelberg, 2001.

[9] R. J. Evey. The Theory and Application of Pushdown Store Machines, Ph.D. Thesis and Research Report, *Mathematics, Linguistics and Automata.* Trans. Project, Harvard University, NSF-10, May 1963.

[10] P. C. Fischer. On Computability by Certain Classes of Restricted Turing machines, In *Proc. Fourth IEEE Symp. on Switch. Circuit Theory and Logical Design*, 1963.

[11] S. Ginsburg. Example of Abstract Machines, *IEEE Trans. on Electronic Computers* 111: 2, 132-135, 1962.

[12] S. Ginsburg and G. F. Rose. Preservation of Languages by Transducers, *Information and Control* 9:2, 153-176, 1966.

[13] O. H. Ibarra. One Two-Way Multihead Automata, *J. Comput. System Sci.* 7, 28-36, 1973.

[14] R. M. Kaplan and M. Kay. Regular Models of Phonological Rule Systems, *Computational Linguistics*, 20(3), 1994.

[15] L. Karttunen. Finite-State Lexicon Compiler. *Technical Report Xerox PARC* P93-00077, Xerox PARC, 1993.

[16] W. Kuich and A. Salomaa. Semirings, Automata, Languages, *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, Germany, 1986.

[17] C. Martín-Vide and V. Mitrana. Parallel Communicating Automata Systems, A Survey. *Korean Journal of Computational Applied Mathematics*, Vol. 7, No. 2, 237-257, 2000.

[18] C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel Finite Automata Systems Communicating by States. *International Journal of Foundation of Computer Science*, Vol. 13 No. 5, 733-749, 2002.

[19] V. Mitrana. On the Degree of Communication in Parallel Communicating Finite Automata Systems. Descriptional Complexity of Automata, Grammars and Related Structures (Magdeburg, 1999). *Journal of Automata Language and Comb.* Vol. 5, No. 3, 301-314, 1999.

[20] M. Mohri. Finite-State Transducers in Language and Speech Processing, *Computational Linguistics*, 23(2), 269-311, 1997.

[21] M. Mohri. Minimization Algorithms for Sequential Transducers, *Theoretical Computer Science*, 234: 177201, 2000.

[22] M. Mohri, F. Pereira, and M. Riley. Weighted Automata in Text and Speech Processing, In *ECAI-96 Workshop*, Budapest, Hungary, 1996.

[23] G. Păun, G. Rozenberg, and A. Salomaa. DNA Computing: new Computing Paradigms, Springer-Verlag, Berlin, Germany, 1998.

[24] D. M. Prescott, A. Ehrenfeucht, and G. Rozenberg. Molecular Operations for Processing in Hypotrichous Ciliates. *European Journal of Protislology*, 2001.

[25] E. Roche and Y. Schabes. *Finite-State Language Processing.* MIT Press, Cambridge, 1997.

[26] G. Rozenberg. Gene Assembly in Ciliates: Computing by Folding and Recombination. In *A half-century of automata theory: celebration and inspiration*, 93-130. World Scientific Publishing, 2001.

[27] M. Sakthi Balan, K. Krithivasan, and M. Madhu. Some Variants in Communication of Parallel Communicating Pushdown Automata. *Journal of Automata, Languages and Combinatorics* 8, 3:401-406, 2003.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 141 - 157, 2004.

Grammar Systems: A Short Survey

Erzsébet Csuhaj-Varjú

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

1 Introduction

"Grammar Systems is a recent field of formal language theory providing syntactic models and frameworks for describing and studying (the behavior of) multi-agent systems at symbolic level." A few years ago, readers interested in the area often found this or similar sentences in papers published about grammar systems. One decade after the publication of the monograph "Grammar systems: A grammatical approach for distribution and cooperation" [24] and after more than fifteen years research in the field, we can state that Grammar Systems has become a well-recognized field of formal language theory.

Several scientific areas have inspired and influenced the developments: 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. However, the relation is two-way: ideas or concepts of grammar systems theory have also inspired and influenced the development of some unconventional computational models in contemporary computer science or research areas being on the boundary of computer science.

But what is a grammar, and what does it mean "system" in this term? The notion grammar refers to any kind of language determining devices, it can be a usual Chomsky grammar, it can be an automaton, but it can also be a splicing system, that is, a rewriting system from molecular computing.

A *grammar system* consists of several (a finite number) of language determining devices (language processors) which jointly change a common symbolic environment (usually, a string or a finite set of strings) by applying language theoretic 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 of time, the state of the system is represented by the current string describing the environment (the collection of strings of the sub-environments). The functioning of the system is realized by changes of its states. Depending on the variant of multi-agent systems which is represented by the actual grammar system, in addition to performing derivation steps,

the language processors communicate with each other. Usually, this is done by exchange of strings which can be data (for example, sentential forms in derivation) or programs (productions or coded form of some operation).

The behavior of the grammar system can be characterized in different manners. It can be represented by the set of sequences of environmental states following each other starting from an initial state or by the set of all states of the environment or that of a sub-environment which originate from the initial state and satisfy certain criteria. The second case defines the language of the system.

According to the traditional approach in formal language theory, one language is generated by one grammar (produced by one language determining mechanism), while according to the non-standard approach provided by Grammar Systems, generation or accepting is performed by several grammars, in cooperation, by a distributed system of language processors. Thus, grammar systems are both syntactic models of multi-agent systems and distributed models of language.

Major problems in the theory can be formulated as follows: to learn whether or not distribution and cooperation enhance the computational power of language processors, to know whether or not grammar systems decrease the complexity of language specification comparing to the language specification given by single grammars, and furthermore, to learn whether or not phenomena characteristic for multi-agent systems (cooperation, distribution, communication, parallelism, emergent behavior, etc.) can be expressed and formalized by tools of grammar systems theory, and if the answer to the last question is positive, which are the proper formalizations.

Although this list of problems is a short one, these items cover a long-long list of subproblems, among them we find questions closely related to basic problems of formal language theory.

To give a picture about the research directions in the area, without the aim of completeness, we list some important frameworks and models and we call the attention to some new problems or research areas which are - according to our opinion - worth studying. The interested reader can find detailed information on grammar systems theory in the monograph [24] and in the book chapter [54], or in the articles listed in the on-line bibliography [43].

For more details concerning the motivations, the background, and connections to the theory of multi-agent systems, artificial intelligence, and artificial life the reader is advised to consult [74, 75, 76, 77, 78, 79].

For possible applications of the theory in linguistics, for connections to natural language generation and modeling the reader is referred to [18, 84, 69, 70, 71].

2 Models, Results, and Problems

2.1 Cooperating Distributed Grammar Systems

Grammar systems theory started in 1988 by introducing cooperating distributed grammar systems (CD grammar systems) for modeling syntactic aspects of the blackboard model of problem solving [22]. The concept and the formal analogy between CD grammar systems and blackboard architectures were discussed and developed further in [27]. Other main inspirations to formalize the concept was the

demand to develop a mathematically well-founded, but sufficiently simple tool for language support of distributed text processing, distributed document creation, and tele-conferencing. We should note, however, that the term "cooperating grammars" was first introduced in [88] as a notion for extending the two-level substitution mechanisms of grammars to a multi-level concept and modeling concurrent operating systems. A notion, related to cooperating grammar systems, called the modular grammar, based on the concept of modularity and motivated by regulated rewriting, was introduced in [4].

A cooperating distributed grammar system is a finite set of (usually generative) grammars which cooperate in deriving words of a common language. At any moment of time, there is exactly one sentential form in generation. The component grammars generate the string by turns, under a cooperation protocol, called the derivation mode. In this model, the cooperating grammars represent independent cooperating problem solving agents which jointly solve a problem by modifying the contents of a global database, called the blackboard which is used for storing information on the problem solving process. In blackboard architectures the agents communicate with each other only through the blackboard, that is, there is no direct communication among them. According to the grammatical framework, the actual contents of blackboard is represented by the sentential form and the generated words correspond to the solutions of the problem.

Most of the investigations in the theory of CD grammar systems have focused on studying the question whether or not cooperation enhances the computational power of the generative capacity of the individual grammars, and, if the answer is positive, how much simple grammars are able to obtain this capacity. Cooperation protocols are decisive factors from this point of view. By the original motivation, the distributed problem solving, most of the studied protocol variants are based on the so-called competence of the grammars, that is, on the capability of the component grammar to perform a derivation step on the actual sentential form. This is usually formalized as the number of different nonterminals (letters) the grammar is able to rewrite in the given string (mainly context-free CD grammar systems have been examined). Most of these protocols are defined via start/stop conditions for the components: these conditions prescribe the competence level of the grammar at starting the derivation and either determine its competence level at the end of its the work or prescribe a number of derivation steps the grammar has to perform in succession (with or without having a certain competence level).

Another important goal of the theory has been to develop protocols where the grammars demonstrate fair behavior, in the sense that their contribution to the generation is almost the same (the number of the performed derivation steps or the number of activation is almost the same for any grammar).

The achieved results demonstrate that CD grammar systems even with simple components and simple protocols form powerful computational devices. Large language classes (the ET0L language class, the class of programmed languages with appearance checking) can be described in terms of systems of a small number of very simple cooperating language determining devices which use relatively very simple, competence-based cooperation protocols, most of them even a protocol requiring fair behavior.

For results about the generative capacity of CD grammar systems using the so-called basic derivation modes, the reader is advised to consult [22, 24, 54, 105, 103]. Other characteristics of the basic, generative model and its different variants have been discussed in various articles. Without the aim of completeness, we mention some of these topics: determinism in CD grammar systems [49, 92], comparisons of systems with components using homogeneous (the same) and hybrid (different for each grammar) cooperation strategies [93, 103, 60, 61], variants of competence-based cooperation protocols [22, 51, 24, 9, 88, 11], similarity of the components [106, 94], hierarchies, priorities, stratification among them [95, 6, 25], fairness in cooperation [46]. CD grammar systems with components working with dynamical start/stop (context) conditions were examined in [23], while CD grammar systems controlled by graphs were considered in [44, 24].

Classical language theoretic questions as descriptional complexity [53, 105] and decidability properties [91] have also been studied.

The original model was introduced for generative grammars, but the idea of the concept has been extended and applied to other computational mechanisms: accepting grammars [58, 59], array grammars [45], limited $0L$ (Lindenmayer) systems [125], automata [48, 82, 32], tree processing devices [63], etc.

Further, powerful extensions of the original notion, as CD grammar systems with registers were studied in [52, 123], and CD grammar systems with communication aided by a generalized sequential machine (a "translator") was the topic of article [124]. Several papers investigated properties of stream X-machines by Eilenberg based on cooperating distributed grammar systems (see [5] for the basic notions and results), and probabilistic CD grammar systems were examined in [3].

The reader can observe that the dominating research areas in the theory of CD grammar systems are the generative capacity and the size complexity of these constructs.

As topics for future research, it would be worth studying the applicability of these systems in solving well-known algorithmic problems, and to investigate cooperation complexity and communication complexity of CD grammar systems. Although steps have already been made to formulate proper notions for cooperation complexity or collective complexity, the problem and the area still needs further elaboration. Similarly the notions of competence and fairness need further elaboration: how these concepts relate to other notions used in the theory of agents or in concurrency theory? Handling incomplete information can also be of particular interest in the theory. Suppose that the problem solving agent has incomplete information on the contents of the blackboard. Has this fact some influence on the problem solving capacity of the problem solving community? This problem can be interpreted in a very nice manner in CD grammar systems theory: some parts of the sentential form are not accessible for the active grammar. How much is the minimal amount of information the agent must have to effectively take part in the problem solving? Another interesting and promising fields of research would be examining CD grammar systems working on multisets of strings and studying how concepts of evolutionary computing can be interpreted in terms of CD grammar systems. Finally, a natural, but yet not explored area is the investigation of the connections between CD grammar systems and games.

While CD grammar systems are sequential computing devices, team grammar systems with simultaneous actions of certain grammars (teams) on the sentential form, introduce parallelism in the model [73]. These systems, among other things, demonstrate the equivalent computational power of programmed grammars with appearance checking and freely chosen grammar teams with a very limited number of components (pairs of grammars) which work with an important variant of competence-based derivations [108]. For further important results and different variants the reader is advised to consult [7, 8, 86].

Team grammar systems provide a wide area of open problems as well. What about the sequence of teams following each other in the course of the generation? Are there problems which can be solved with teams working periodically, that is, with periodical sequences of teams that follow each other in the course of the derivation? What about problems where these team sequences are aperiodical? What about the dynamism of the size, the simplicity of the components, and behavior of teams that are formed according to the actual competence of the grammars? How concepts of evolutionary computing can be implemented in this area? And, last but not least a general question: are team grammar systems more effective in problem solving than CD grammar systems (for example, according to their derivation complexity) and, if this is the case, when?

2.2 Colonies and Eco-Grammar Systems

Colonies are important concepts in grammar systems theory that describe language classes in terms of collections of very simple, purely reactive, situated agents with emergent behavior [80, 81]. In this model the agents are represented by very simple regular grammars (each grammar generates a finite language) which generate a common sentential form. The basic variant of colonies determines the context-free language class, while the more sophisticated models (competition among the agents, agents with point mutation operations) represent computational tools with considerably enhanced computational power [47, 87]. The interested reader can find several other interesting models and results in the articles referred in the bibliography [43].

For future research, further elaboration of notion of emergence in terms of formal language theory would be fundamental in the area. In addition, this field appears to be particularly suitable for finding applications, especially in robotics.

Eco-grammar systems form a language theoretic framework for modeling ecosystems: developing agents, represented by L systems, in a dynamically changing population, interact with each other and with their shared evolving symbolic environment (See [28] and [29] for basic information). These constructions were motivated by artificial life and provide tools for describing life-like phenomena (birth, death, hibernation, overpopulation, pollution, etc.) in terms of formal grammars and languages. For detailed information on the results obtained in this area and in the theory of colonies the reader to the articles listed in the on-line annotated bibliography [43]. While according to the original concept of eco-grammar systems the agents perform local actions on the environment, so-called conditional tabled eco-grammar systems are models with indirect interaction between the agents and the environment. This subfield has been extensively explored, see [33, 34, 114, 115, 116, 117] for further

information.

Eco-grammar systems are very promising from the point of view of future investigations. Firstly, the relation of the different variants of eco-grammar systems and that of evolutionary computing are of worth analyzing, but the general concept of eco-grammar systems can be related to several other scientific areas: modeling economy, modeling society, modeling the World Wide Web. Valuable research has started in modeling culture and pragmatics in [70] and investigations of the relations between dynamical systems and eco-grammar systems have been started in [14].

2.3 Network of language processors

Networks of language processors are one of the basic areas in the theory of grammar systems. In these constructs, language processors, that is, grammars or other language determining devices 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 which can be data and/or programs. Rewriting and communication take place alternately. The system functions (usually) in a synchronized manner.

It is easy to observe the difference between CD grammar systems and these architectures: while in the first case the grammars generate one common string, in the second case each of the components operates on its own string. There are several important different models in the area, "networks of language processors" as a collective term was introduced in [36, 19, 37], where basic characteristics and basic variants of the framework were discussed and several research directions were proposed.

Parallel communicating grammar systems (or PC grammar systems, for short), the first models in the area of networks of language processors, form a very important, highly elaborated field, according to the original definition with Chomsky grammars at the nodes. These are networks of grammars with components communicating strings by emerging requests.

The concept was introduced in [110] (and continued in [111, 99]), as a grammatical representation of the so-called " classroom model" of problem solving, which is a modification of the blackboard architecture. It consists of several agents which jointly solve a problem. Each agent has its own "notebook" containing the description of a particular subproblem of the given problem and each agent operates only on its own "notebook" and one distinguished agent, the master, operates on the "blackboard". This agent has the description of the whole problem and it decides when the problem is solved. Moreover, the agents communicate with each other by requests concerning the contents of their "notebooks." According to this model, the agents can be considered as pupils in a classroom and the master is their classroom leader or teacher.

According to the grammatical model, each agent is represented by a grammar which generates its own sentential form. In each time unit, every grammar performs a rewriting step on its own sentential form (modifies its own notebook) and communication is done by requests through so-called query symbols, one different symbol referring to each grammar in the system. When a query symbol appears in the sen-

tential form of a grammar, then the rewriting process stops and one or more communication steps are performed by replacing all occurrences of the query symbols with the current sentential forms of the queried component grammars, supposing that these strings are query-free. When no more query symbol is present in any of the sentential forms, then the rewriting process starts again. In so-called returning systems after communicating its current sentential form the queried component returns to its start symbol and begins to generate a new string. In non-returning systems the components continue the rewriting of their sentential form. The language (the set of problem solutions) of the system is the set of terminal words which appear as sentential forms at the master.

Parallel communicating grammar systems, as cooperating distributed grammar systems, have been the subject of detailed study during the years. The investigations concentrated on the generative power of these systems and on examining how this power is influenced by changes in the basic characteristics of the system: the way of communication and synchronization among the components and their way of functioning. The results, as in the case of CD grammar systems, demonstrate that PC grammar systems with very simple component grammars and with bounded size are convenient tools for describing large language classes. For example, non-context-free context-sensitive languages can be generated by PC grammar systems with two regular components, and returning PC grammar systems with a few number of context-free component grammars are sufficient to generate any recursively enumerable language. For information concerning the generative power of PC grammar systems the reader is referred to [24, 54] and the articles listed in [43]. For details about the generative capacity and size of systems with regular or right-linear components consult [110, 122, 72], results on systems with context-free components are presented in [39, 83], and systems with context-sensitive components are discussed in [113, 62].

Normal forms for context-free PC grammar systems are presented in [40, 42], and normal forms for systems with right-linear components are given in [56].

The very important feature of context-free (linear, regular) PC grammar systems that the returning systems simulate the non-returning systems was pointed out in [55, 120]. Synchronization problems, that is, PC grammar systems with additional synchronization mechanisms or unsynchronized systems, were studied in [98, 102, 40]. Different ways of communication and their impact on the generative power were examined in [40], and different ways of defining the language of the system (so-called popular and competitive systems) were proposed and discussed in [119]. An important aspect, namely the case of incomplete information communication, where the grammars communicate subwords of their sentential form by request, was studied in [41].

Communication complexity problems of PC grammar systems were investigated in [96, 97, 64, 65, 66, 68], their computational complexity aspects were discussed in [1, 16, 15], while results concerning the descriptional complexity of these constructs were presented in [100, 24]. Some decision problems concerning PC grammar systems were examined in [118].

Several other interesting variants were proposed and investigated: PC grammar systems with query words [89], with separated alphabets [90], with signals [112],

with negotiations [107], probabilistic PC grammar systems [2], etc.

An area, with distinguished importance is the theory of parallel communicating L systems, started in [101]. Several interesting results were obtained in this field, among other things it was shown that centralized PC Lindenmayer systems with $D0L$ components can be simulated with $EDT0L$ systems [101], and that systems with $E0L$ and $ET0L$ components are equally powerful both in the returning and in the non-returning case, [121]. For detailed information, the reader should consult [24] and the articles in the on-line annotated bibliography [43].

As in the case of CD grammar systems, the concept of PC grammar systems has been extended to parallel communicating automata systems. For the finite automata models we refer to [85], for basic results on parallel communicating pushdown automata systems consult [31].

For future research, interesting an important topics in the theory of PC grammar systems can be studying PC grammar systems with asynchronously working components, and further variants with incomplete information communication. For example, models where the grammars are allowed to communicate only subwords with a certain fixed length or with a bounded length, or where the grammars split the sentential form into as many pieces as were requested by the other grammars. Another promising directions can be the implementations of well-known distributed algorithms in this framework and the comparison of the efficiency of the different methods.

While parallel communicating grammar systems form a network of Chomsky grammars communicating by the dynamically emerging requests, CCPC grammar systems, that is, parallel communicating grammar systems with communication by command, represent another communication philosophy. In this case, after each rewriting step (after performing a prescribed sequence of rewriting steps) the Chomsky grammars communicate with each other by exchanging information which is represented by their current sentential form or a subword of it. The communicated strings are filtered: each component is associated with a so-called selector (filter) language. A communicated string is accepted by the component if it is an element of its selector language. After communication, a new string is composed from the accepted ones and the grammar begins to generate this word. For more details about the motivations beyond the idea and for the possible different variants the reader is referred to [30]. A basic and surprising result concerning these constructions is, that CCPC grammar systems with three regular grammars and regular filter languages are able to generate any recursively enumerable language [67].

Similar architectures, that is, networks of language processors communicating by command with the aid of filter languages were studied having $0L$ systems as components in [36], called networks of parallel language processors. Test tube distributed systems based on splicing [26] and networks of Watson-Crick $D0L$ systems [21, 38] are also examples for networks of language processors communicating by command, with components inspired by the behavior and properties of DNA, namely DNA recombination and the phenomenon of Watson-Crick complementarity. Investigations in a framework, called networks of evolutionary processors, where the language determining devices are simple rewriting systems with point mutation operations, have recently started with interesting results. (See for the basic notions [17]).

According to these latter, bio-inspired constructs, promising and challenging research areas can be studying the population dynamics of the objects (strings, symbols, etc.) in the network, supposing that the language processors operate with multisets of strings. Phenomena characteristic for real networks, namely overloaded situations at the nodes, black-holes, chaotic-like events, and wave-like phenomena would also worth formalizing and studying.

Since networks of language processors are closely related to membrane systems, bridges between membrane systems theory, that is, P systems theory should also be built.

3 Conclusions

Grammar Systems is a wide and rich area for further investigations, both concerning classical problems of formal language theory and non-standard problems as descriptions of multi-agent systems in terms of distributed systems of grammars. One of the most interesting and promising problem area would be the study of the properties of grammar systems with dynamically changing (evolving, adapting) components. Similarly, open networks of language processors, that is, networks with dynamically changing number of components would be worth studying. A promising research area is the study of grammar systems operating over multisets of strings, especially in the direction of connections to molecular computing, in particular membrane computing [104]. Another challenging field for research is to find connections between game theory and grammar systems theory, especially in the case of CD grammar systems. Obviously, classical language theoretic problems, that is, decidability questions, communication complexity and computational complexity issues are of interest. Grammar Systems theory opens a new research area in descriptional complexity as well, namely it gives the possibility to develop the concept of "collective complexity" and to compare the behavior of the corresponding measures to the behavior of the well-known complexity measures based on single grammars in the case of different language classes.

References

[1] K. Abrahamson, L. Cai, and S. Gordon. A grammar characterization of logarithmic space computation. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Lecture Notes in Computer Science*, pages 147–155. Springer-Verlag, 1997.

[2] K. Arthi and K. Krithivasan. Probabilistic parallel communicating grammar systems. *International Journal of Computer Mathematics*, 79(1):1–26, 2002.

[3] K. Arthi, K. Krithivasan and E. Csuhaj-Varjú. On rule-number complexity of components of probabilistic cooperating distributed grammar systems. *Journal of Automata, Languages and Combinatorics*, 7(4):433–446, 2002.

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

[5] T. Bălănescu, T. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, C. Vertan. Communicating Stream X-machines Systems are no more than X-machines. *Journal of Universal Computer Science*, 5(9):494–507, 1995.

[6] T. Bălănescu, H. Georgescu and M. Gheorghe. Grammar systems with counting derivation and dynamical priorities. In Gh. Păun and A. Salomaa editors, *New Trends in Formal Languages. Control, Cooperation and Combinatorics*, volume 1218 of Lecture Notes in Computer Science, pages 150–166. Springer, Berlin, 1997.

[7] M. H. ter Beek. Teams in grammar systems: hybridity and weak rewriting. *Acta Cybernetica*, 12(4):427–444, 1996.

[8] M. H. ter Beek. Teams in grammar systems: sub-context-free cases. In Gh. Păun and A. Salomaa editors, *New Trends in Formal Languages. Control, Cooperation and Combinatorics*, volume 1218 of Lecture Notes in Computer Science, pages 197–216. Springer, Berlin, 1997.

[9] H. Bordihn and E. Csuhaj-Varjú. On competence and completeness in CD grammar systems. *Acta Cybernetica*, 12(4):347–360, 1996.

[10] H. Bordihn, E. Csuhaj-Varjú, and J. Dassow. CD grammar systems versus L systems. In Gh. Păun and A. Salomaa editors, *Grammatical models of multi-agent systems*, pages 18–32. Gordon and Breach, Amsterdam, 1999.

[11] H. Bordihn and M. Holzer. Grammar systems with negated conditions in their cooperation protocols. *Journal of Universal Computer Science* 6(12):1165-1184, 2000.

[12] H. Bordihn and M. Holzer. On the Number of Active symbols in L and CD Grammar Systems. *Journal of Automata, Languages and Combinatorics* 6(4): 411-426, 2001.

[13] H. Bordihn and B. Reichel. On descriptions of context-free languages by CD grammar systems. *Journal of Automata, Languages and Combinatorics*, 7(4):447–454, 2002.

[14] B. Cases and M. Alfonseca. Eco-grammar systems applied to the analysis of the dynamics of population in artificial worlds. In R. Freund and A. Kelemenová editors, *Proceedings of the International Workshop Grammar Systems 2000*, pages 265–281. Silesian University at Opava, 2000.

[15] L. Cai. Computational completeness of linear PCGSs. *Computers and Artificial Intelligence*, 15(2-3):199–210, 1996.

[16] L. Cai. The computational complexity of PCGS with regular components. In *Proceedings of the 2nd International Conference on Developments in Language Theory*, pages 209–219, Singapore, 1996. World Scientific Publishers.

[17] J. Castellanos, C. Martín-Vide, V. Mitrana and J. Sempere. Networks of evolutionary processors. *Acta Informatica*, 39(6-7):517–529, 2003.

[18] E. Csuhaj-Varjú. Grammar systems: A framework for natural language generation. In Gh. Păun editor, *Mathematical Aspects of Natural and Formal Languages*, *World Scientific Series in Computer Science 43*, pages 63–78. World Scientific, Singapore, 1994.

[19] E. Csuhaj-Varjú. Networks of language processors. *EATCS Bulletin*, 63:120–134, 1997.

[20] E. Csuhaj-Varjú. On size complexity of context-free returning parallel communicating grammar systems. In C. Martin-Vide and V. Mitrana editors, *Where Mathematics, Computer Science, Linguistics and Biology Meet.* pages 37–49. Kluwer Academic Publishers, Dordrecht, 2001.

[21] E. Csuhaj-Varjú. Computing by networks of standard Watson-Crick *D0L* systems. communicating grammar systems. In M. Ito editor, *Proc. Workshop Algebraic Systems, Languages and Computations, March 21-23, 2000, Kyoto*, pages 43–51. RIMS Kokyuroku 1116, 2000.

[22] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics EIK*, 26(1-2):49–63, 1990. Presented at the "4th Workshop on Mathematical Aspects of Computer Science", Magdeburg, 1988.

[23] E. Csuhaj-Varjú, J. Dassow and Gh. Păun. Dynamically controlled cooperating/distributed grammar systems. *Information Sciences*, 69(1-2):1–25, 1993.

[24] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun. *Grammar systems. A grammatical approach to distribution and cooperation. Topics in Computer Mathematics 8.* Gordon and Breach Science Publishers, Yverdon, 1994.

[25] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun. Stratified grammar systems. *Computers and Artificial Intelligence*, 13(5):409–422,

[26] E. Csuhaj-Varjú, L. Kari, and Gh. Păun. Test tube distributed systems based on splicing. *Computers and Artificial Intelligence*, 15(2-3):211–232, 1996.

[27] E. Csuhaj-Varjú and J. Kelemen. Cooperating grammar systems: A syntactical framework for blackboard model of problem solving. In I. Plander, editor, *Proceedings of the Conference on Artificial Intelligence and Information Control System of Robots, AIICSR'89*, pages 121–127. Elsevier Publishing Company, Amsterdam, 1989.

[28] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, and Gh. Păun. Eco(grammar) Systems. A Preview. In R. Trappl, editor, *Proc 12th European Meeting on Cybernetics and System Research*, pages 941–948. World Scientific Publishers, Singapore, 1994.

[29] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, and Gh. Păun. Eco-Grammar Systems: A Grammatical Framework for Studying Lifelike Interactions. *Artificial Life*, 3:1–28, 1997.

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

[31] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and Gy. Vaszil. Parallel communicating pushdown automata systems. *International Journal of Foundations of Computer Science*, 11(4):633–650, 2000.

[32] E. Csuhaj-Varjú, V. Mitrana, and Gy. Vaszil. Distributed Pushdown Automata Systems: Computational Power. In Z. Ésik and Z. Fülöp, editors, *Developments in Language Theory. 7th International Conference.* , volume 2710 of *Lecture Notes in Computer Science*, pages 218–229. Springer-Verlag, 2003.

[33] E. Csuhaj-Varjú, Gh. Păun and A. Salomaa. Conditional Tabled Eco-Grammar Systems. In Gh. Păun editor, *Artificial Life: Grammatical Models*, pages 227–239. Black Sea University Press, Bucharest, 1995.

[34] E. Csuhaj-Varjú, Gh. Păun, and A. Salomaa. Conditional Tabled Eco-Grammar Systems versus (E)T0L Systems. *Journal of Universal Computer Science*, 5(1):252–268, 1995.

[35] E. Csuhaj-Varjú, Gh. Păun and Gy. Vaszil. PC grammar systems with five context-free components generate all recursively enumerable languages. (Note) *Theoretical Computer Science*, 299:785-794, 2003.

[36] E. Csuhaj-Varjú and A. Salomaa. Networks of parallel language processors. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages; Control, Cooperation, and Combinatorics*, volume 1218 of *Lecture Notes in Computer Science*, pages 299–318. Springer-Verlag, 1997.

[37] E. Csuhaj-Varjú and A. Salomaa. Networks of language processors: Parallel communicating systems. *EATCS Bulletin*, 66:122–138, 1998.

[38] E. Csuhaj-Varjú and A. Salomaa. Networks of Watson-Crick $D0L$ systems. In M. Ito and T. Imaoka, editors, *Words, Languages & Combinatorics III*, World Scientific, 2003.

[39] E. Csuhaj-Varjú and Gy. Vaszil. On the computational completeness of context-free parallel communicating grammar systems. *Theoretical Computer Science*, 215(1-2):349–358, 1999.

[40] E. Csuhaj-Varjú and Gy. Vaszil. On context-free parallel communicating grammar systems: Synchronization, communication, and normal forms. *Theoretical Computer Science*, 255: 511–538, 2001.

[41] E. Csuhaj-Varjú and Gy. Vaszil. Parallel Communicating Grammar Systems with Incomplete Information Communication. *Grammars*, 5: 153–176, 2002.

[42] E. Csuhaj-Varjú and Gy. Vaszil. Parallel communicating grammar systems with bounded resources. *Theoretical Computer Science*, 276: 205–219, 2002.

[43] E. Csuhaj-Varjú and Gy. Vaszil. An annotated bibliography of grammar systems. http://www.sztaki.hu/mms/bib.html

[44] J. Dassow. A remark on cooperating distributed grammar systems controlled by graphs. *Wissenschaftliche Zeitung T.U. Magdeburg*, 35:4–6, 1991.

[45] J. Dassow, R. Freund, and Gh. Păun. Cooperating array grammar systems. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(6):1029–1053, 1995.

[46] J. Dassow and V. Mitrana. Fairness in grammar systems. *Acta Cybernetica*, 12(4):331–345, 1996.

[47] J. Dassow, J. Kelemen and Gh. Păun. On parallelism in colonies. *Cybernetics and Systems: An International Journal*, 24(1):37–49, 1993.

[48] J. Dassow and V. Mitrana. Stack cooperation in multi-stack pushdown automata. *Journal of Computer and System Sciences* 58:611–621, 1999.

[49] J. Dassow, V. Mitrana, Gh. Păun and S. Vicolov. On determinism in cooperating distributed grammar systems. *Publicationes Mathematicae* 48(3-4):217–231, 1996.

[50] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Springer Verlag, Berlin, 1989.

[51] J. Dassow and Gh. Păun. On some variants of cooperating/distributed grammar systems. *Studii şi Cerecetări Matematice*, 42:153-165, 1990.

[52] J. Dassow and Gh. Păun. Cooperating/distributed grammar systems with registers. *Foundations of Control Engineering*, 15(1):19–38, 1990.

[53] J. Dassow and Gh. Păun. On the succinctness of descriptions of context-free languages by cooperating/distributed grammar systems. *Computers and Artificial Intelligence*, 10(6):513–527, 1991.

[54] J. Dassow, Gh. Păun, and G. Rozenberg. Grammar systems. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 2, chapter 4, pages 155–213. Springer-Verlag, Berlin-Heidelberg, 1997.

[55] S. Dumitrescu. Non-returning PC grammar systems can be simulated by returning systems. *Theoretical Computer Science*, 165:463–474, 1996.

[56] S. Dumitrescu and Gh. Păun. On the power of parallel communicating grammar systems with right-linear components. *RAIRO Informatique Theoretique et Applications*, 31(4):331–354, 1997.

[57] S. Dumitrescu, Gh. Păun, and A. Salomaa. Pattern languages versus parallel communicating grammar systems. *International Journal of Foundations of Computer Science*, 8(1):67–80, 1997.

[58] H. Fernau and M. Holzer. Accepting multi-agent systems. II. *Acta Cybernetica*, 12(4):361–379, 1996.

[59] 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.

[60] H. Fernau, M. Holzer and R. Freund. Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theoretical Computer Science*, 259: 405-426, 2001.

[61] H. Fernau, M. Holzer and R. Freund. Hybrid modes in cooperating distributed grammar systems: combining the t-mode with the modes $\leq k$ and $= k$. *Theoretical Computer Science*, 299(1-3): 633-662, 2003.

[62] R. Freund, Gh. Păun, C. M. Procopiuc, and O. Procopiuc. Parallel communicating grammar systems with context-sensitive components. In Gh. Păun, editor, *Artificial Life. Grammatical Models*, pages 166–174. The Black Sea University Press, Bucharest, 1995.

[63] Z. Fülöp. Distributed tree processing devices. *Computers and Artificial Intelligence*, 18(1):37–71, 1999.

[64] J. Hromkovič. On the communication complexity of distributed language generation. In J. Dassow, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory II*, pages 237–246. World Scientific Publishers, 1996.

[65] J. Hromkovič, J. Kari, and L. Kari. Some hierarchies for the communication complexity measures of cooperating grammar systems. *Theoretical Computer Science*, 127:123–147, 1993.

[66] J. Hromkovič, J. Kari, L. Kari, and D. Pardubská. Two lower bounds on distributive generation of languages. In *MFCS'94*, volume 841 of *Lecture Notes in Computer Science*, pages 423–432. Springer-Verlag, 1994.

[67] L. Ilie and A. Salomaa. 2-testability and relabelings produce everything. *Journal of Computer and System Sciences*, 56(3):253–262, 1998.

[68] C. M. Ionescu and O. Procopiuc. Bounded communication in parallel communicating grammar systems. *Journal of Information Processing and Cybernetics*, 30(2):97–110, 1994.

[69] M. D. Jiménez-López and C. Martín-Vide. Grammar systems for the description of certain natural language facts. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages; Control, Cooperation and Combinatorics*, volume 1218 of *Lecture Notes in Computer Science*, pages 288–298. Springer-Verlag, 1997.

[70] M. D. Jiménez-López. Grammar Systems: A Formal-Language-Theoretic Framework for Linguistics and Cultural Evolution. Ph. D. Dissertation, Rovira i Virgili University, Tarragona, 2000.

[71] M. D. Jiménez-López. Linguistic Grammar Systems: A Grammar Systems Approach to Natural Languages. In C. Martín-Vide and V. Mitrana editors, *Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology, and Back*, pages 55–65. Taylor and Francis, London, 2003. Rovira i Virgili University, Tarragona, 2000.

[72] J. Kari and L. Sântean. The impact of the number of cooperating grammars on the generative power. *Theoretical Computer Science*, 98:621–633, 1992.

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

[74] J. Kelemen. Syntactic models of distributed cooperative systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 3:1–10, 1991.

[75] J. Kelemen. Grammar-theoretic models of multi-agent systems. *Kybernetes*, 23(2):43–50, 1994.

[76] J. Kelemen. Artificial life-describing life-like behaviors in computational frameworks. In Gh. Păun, editor, *Artificial Life. Grammatical Models*, pages 1–21. The Black Sea University Press, Bucharest, 1995.

[77] J. Kelemen. Colonies as models of reactive systems. In Gh. Păun and A. Salomaa editors, *New Trends in Formal Languages. Control, Cooperation and Combinatorics*, volume 1218 of Lecture Notes in Computer Science, pages 220–235. Springer, Berlin, 1997.

[78] J. Kelemen. On post-modularity and emergence from grammar-theoretic point of view. In P. Sincak and J. Vascak editors, *Quo Vadis Computational Intelligence?*, pages 342–352. Physica-Verlag, Heidelberg, 2000.

[79] J. Kelemen. From statistics to emergence–exercises in systems modularity. In M. Luck et al. editors, *Multi-Agent Systems and Applications*, volume 2086 of Lecture Notes in Computer Science, pages 281–300. Springer, Berlin, 2001.

[80] J. Kelemen and A. Kelemenová. A subsumption architecture for generative symbol systems. In R. Trappl editor, *Cybernetics and Systems'92*, pages 1529-1536. World Scientific, Singapore, 1992.

[81] J. Kelemen and A. Kelemenová. A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems: An International Journal*, 23(6):621–633, 1992.

[82] K. Krithivasan, M. Sakthi Balan and P. Harsha. Distributed Processing in Automata. *International Journal of Foundations of Computer Science*, 10(4):443-464, 1999.

[83] N. Mandache. On the computational power of context-free PC grammar systems. *Theoretical Computer Science*, 237(1-2):135–148, 2000.

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

[85] C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel finite automata systems communicating by states. *International Journal of Foundations of Computer Science*, 12(5):733-749, 2002.

[86] A. Mateescu, V. Mitrana, and A. Salomaa. Dynamical teams of cooperating grammar systems. *Annales Univ. Bucureşti Mat. Inform.*, 42(43)(Matematica-Informatica):3–14, 1993/94.

[87] Carlos Martín-Vide and Gheorghe Păun. PM-colonies. *Computers and Artificial Intelligence*, 17(6):553–582, 1998.

[88] R. Meersman and G. Rozenberg. Cooperating grammar systems. In J. Winkowski editor, *Proceedings of Mathematical Foundations of Computer Science, 1978*, volume 64 of Lecture Notes in Computer Science, pages 364–373. Springer, Berlin-New York, 1978.

[89] V. Mihalache. Parallel communicating grammar systems with query words. *Analele Universităţii Bucureşti. Matematică-Informatică*, 45(1):81–92, 1996.

[90] V. Mihalache. Parallel communicating grammar systems with separated alphabets. *Acta Cybernetica*, 12(4):397–409, 1996.

[91] V. Mihalache. Decidability problems in grammar systems. *Theoretical Computer Science*, 215(1-2):169–189, 1999.

[92] V. Mihalache and V. Mitrana. Deterministic cooperating distributed grammar systems. In Gh. Păun and A. Salomaa editors, *New Trends in Formal Languages. Control, Cooperation and Combinatorics*, volume 1218 of Lecture Notes in Computer Science, pages 137–149. Springer, Berlin, 1997.

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

[94] V. Mitrana. Similarity in grammar systems. *Fundamenta Informaticae*, 24(3):251–257, 1995.

[95] V. Mitrana, Gh. Păun and G. Rozenberg. Structuring grammar systems by priorities and hierarchies. *Acta Cybernetica*, 11(3):189–204, 1994.

[96] D. Pardubská. The communication complexity hierarchy of parallel communicating grammar systems. In J. Dassow and A. Kelemenová, editors, *Developments in Theoretical Computer Science*, pages 115–122. Gordon and Breach Science Publishers, London, 1992.

[97] D. Pardubská. On the power of communication structure for distributive generation of languages. In G. Rozenberg and A. Salomaa, editors, *Developments in Language Theory*, pages 419–429. World Scientific Publishers, 1994.

[98] Gh. Păun. On the power of synchronization in parallel communicating grammar systems. *Studii şi Cercetări Matematice*, 41(3):191–197, 1989.

[99] Gh. Păun. Parallel communicating grammar systems: The context-free case. *Foundations of Control Engineering*, 14(1):39–50, 1989.

[100] Gh. Păun. On the syntactic complexity of parallel communicating grammar systems. *Kybernetika*, 28(2):155–166, 1992.

[101] Gh. Păun. Parallel communicating systems of L systems. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, pages 405–418. Springer Verlag, 1992.

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

[103] Gh. Păun. On the generative capacity of hybrid CD grammar systems. *Journal of Information Processing and Cybernetics EIK*, 30(4):231–244, 1994.

[104] Gh. Păun. *Membrane Computing. An Introduction.*, Springer, 2002.

[105] Gh. Păun, J. Dassow and S. Skalla. On the size of components of cooperating grammar systems. In J. Karhumäki, H.A. Maurer and G. Rozenberg editors, *Results and Trends in Theoretical Computer Science*, volume 812 of *Lecture Notes in Computer Science*, pages 325–343. Springer, Berlin, 1994.

[106] G. Păun, V. Mitrana, E. Csuhaj-Varjú and J. Dassow. Cooperation in grammar systems: similarity, universality, timing. *Cybernetica*, 36(4):271–286, 1993.

[107] Gh. Păun, L. Polkowski, and A. Skowron. Parallel communicating grammar systems with negotiation. *Fundamenta Informaticae*, 28(3-4):315–330, 1996.

[108] Gh. Păun and G. Rozenberg. Prescribed teams of grammars. *Acta Informatica*, 31(6):525–537, 1994.

[109] Gh. Păun, A. Salomaa, and S. Vicolov. On the generative capacity of parallel communicating grammar systems. *International Journal of Computer Mathematics*, 45:45–59, 1992.

[110] Gh. Păun and L. Sântean. Parallel communicating grammar systems: The regular case. *Annals of the University of Bucharest, Mathematics-Informatics Series*, 38(2):55–63, 1989.

[111] Gh. Păun and L. Sântean. Further remarks on parallel communicating grammar systems. *International Journal of Computer Mathematics*, 34:187–203, 1990.

[112] D. Popescu. Parallel communicating grammar systems with communication by signals. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages; Control, Cooperation, and Combinatorics*, volume 1218 of *Lecture Notes in Computer Science*, pages 267–277. Springer-Verlag, Berlin, 1997.

[113] O. Procopiuc, C. M. Ionescu, and F. L. Ţiplea. Parallel communicating grammar systems: The context-sensitive case. *International Journal of Computer Mathematics*, 49:145–156, 1993.

[114] P. Sosík. Eco-Grammar Systems, Decidability and the Tiling Problem. In A. Kelemenová editor, *Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems*, pages 195–213. Silesian University, Opava, 1998.

[115] P. Sosik. On the Hierarchy of Extended Conditional Tabled Eco-Grammar Systems. *Grammars*, 1(3): 225–238,1999.

[116] P. Sosik On the Decidability Problems of Eco-Grammar Systems. *Journal of Automata, Languages and Combinatorics*, 5(1): 45–58.

[117] P. Sosík. Conditional tabled eco-grammar systems and random context grammars. In R. Freund and A. Kelemenová editors, *Proceedings of the International Workshop Grammar Systems 2000*, pages 297–308. Silesian University at Opava, 2000.

[118] F. L. Ţiplea, C. Ene, C. M. Ionescu, and O. Procopiuc. Some decision problems for parallel communicating grammar systems. *Theoretical Computer Science*, 134:365–385, 1994.

[119] Gy. Vaszil. Parallel communicating grammar systems without a master. *Computers and Artificial Intelligence*, 15(2-3):185–198, 1996.

[120] Gy. Vaszil. On simulating non-returning PC grammar systems with returning systems. *Theoretical Computer Science*, 209:319–329, 1998.

[121] Gy. Vaszil. On parallel communicating Lindenmayer systems. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, volume 8 of *Topics in Computer Mathematics*, pages 99–112. Gordon and Breach Science Publishers, 1999.

[122] S. Vicolov. Non-centralized parallel communicating grammar systems. *Studii şi Cercetări Matematice. Mathematical Reports*, 44(5):455–462, 1992.

[123] S. Vicolov. Cooperating/distributed grammar systems with registers: the regular case. *Computers and Artificial Intelligence*, 12(1):89–98, 1993.

[124] S. Vicolov-Dumitrescu. Grammars, grammar systems, and gsm mappings with valences. In Gh. Păun editor, *Mathematical Aspects of Natural and Formal Languages*, *World Scientific Series in Computer Science 43*, pages 473–491. World Scientific, Singapore, 1994.

[125] D. Wätjen. On cooperating/distributed limited 0L systems. *Journal of Information Processing and Cybernetics EIK*, 29:129–142, 1993.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 158 - 169, 2004.

On Cooperating Distributed Grammar Systems with Competence Based Start and Stop Conditions

Jürgen Dassow

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
PSF 4120, D–39016 Magdeburg
dassow@iws.cs.uni-magdeburg.de

Abstract

We define cooperating distributed grammar systems with start and stop conditions which are based on the competence of a component on the current sentential form. We distinguish six different types of competence conditions which result in 18 types of grammar systems. We summarize the results on the generative power known from the literature (where they are sometimes not related to competence) and determine the power of some further grammar systems.

1 Introduction

Cooperating distributed grammar systems were firstly investigated by R. Meersman and G. Rozenberg in [14]. A systematic study of these systems was started in [6]. Summaries of results are given in [9] and [11].

Intuitively, a cooperating distributed grammar system (CD grammar system for short) consists of some grammars or sets of productions which work on a common sentential form. A certain grammar, for which the start condition holds, starts the derivation, and it has to stop the derivation, if a certain stop condition is satisfied. Then another component satisfying the start condition continues the derivation until the stop condition holds etc.

Mostly one has considered the case where the start condition is true for any grammar and any sentential form and the stop condition is satisfied iff the derived sentential form y is obtained by k (or $\geq k$ or $\leq k$) direct derivation steps with respect to the chosen grammar or the grammar contains no rule which can be applied to y.

The idea behind these conditions was as follows. CD grammar systems have a motivation in the blackboard architecture of Artificial Intelligence. Here the grammars correspond to agents/experts, the nonterminals represent open problems and the application of a rule is a step to the solution. Thus the above conditions can be interpreted such that k (or $\geq k$ or $\leq k$) steps can be contributed to the solution. Therefore the stop condition is only satisfied if the component has a certain competence. However, this does not really reflect competence, because in the k-step derivation we can replace the same nonterminal in any step, i.e., we contribute

only to a special subproblem and not to all subproblems. Therefore the condition considered in [14] is more appropriate, where the start condition requires that any nonterminal occurring in the sentential form can be replaced, i.e., the agent can contribute to any subproblem; we call this restriction full competence of the grammar on the sentential form.

In this paper we define the competence as the number of subproblems which can be (partially) solved by the grammar. Formally, the competence of a set P of productions on a sentential form x is the cardinality of the intersection of the set of nonterminals occurring in x and the set of nonterminals which can be rewritten by P. Now one can define competence k (or $\geq k$ or $\leq k$ or $\neq k$) and maximal and full competence. This leads to 18 types of CD grammar systems where the start condition as well as the stop condition is one of the competence types mentioned above or a negation of such a competence type.

We summarize the results on the generative power of such CD grammar systems obtained by M. ter Beek, E. Csuhaj-Varjú, M. Holzer, Gy. Vaszil and the author and add the results on two further systems.

The paper is organized as follows. In Section 2 we recall the definitions of some language families to which the families studied in the paper will be related. In Section 3 we introduce the concept of a cooperating distributed grammar system and the competence conditions. Section 4 contains the results. We finish with some remarks on topics of research to be done in this area.

2 Some Language Families

For an alphabet V, we denote the set of all (non-empty) words over V by V^* (and V^+, respectively). The length of a word $w \in V^*$ is denoted by $|w|$. For a letter $a \in V$ and a word $w \in V^*$, $\#_a(w)$ denotes the number of occurrences of a in w.

A *context-free grammar* is specified as a quadruple $G = (N, T, P, S)$ where
- N and T are disjoint alphabets of nonterminals and terminals, respectively,
- P is a finite subset of $N \times (N \cup T)^*$, and
- S is an element of N.

Instead of (A, w) for an element of P, we shall write $A \to w$. Elements of P are called context-free rules. We set $V_G = N \cup T$. The derivation process in a context-free grammar and the generated language are defined as usually (see e.g. [17]).

A *Russian parallel grammar* is a quintuple $G = (N, T, P_1, P_2, S)$, where
- N, T, and S are specified as in a context-free grammar, and
- P_1 and P_2 are finite sets of $N \times (N \cup T)^*$.

$x \in V_G^+$ directly derives $y \in V_G^*$ (written as $x \Longrightarrow y$), iff one of the following conditions hold:
- $x = x'Ax''$, $y = x'wx''$ for some $x', x'' \in V_G^*$ and $A \to w \in P_1$ or
- $x = x_0Ax_1Ax_2 \ldots x_{n-1}Ax_n$, $n \geq 0$, $x_i \in (V_G \setminus \{A\})^*$ for $1 \leq i \leq n$, $A \to w \in P_2$

and $y = x_0wx_1wx_2 \ldots x_{n-1}wx_n$.

The language $L(G)$ generated by the Russian parallel grammar G is defined as

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

where $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$.

A *random context grammar* is a quadruple $G = (N, T, P, S)$ where
– N, T and S are specified as in a context-free grammar, and
– P is a finite set of triples $r = (p, R, Q)$ where p is a context-free production and R and Q are subsets of N.
G is called a *forbidden* random context grammar if all rules of P are of the form $(p, \emptyset, Q)$. For $x, y \in V_G^*$, we say that x directly derives y, written as $x \Longrightarrow y$, iff there is a triple $r = (A \to w, R, Q) \in P$ such that
– $x = x'Ax''$ and $y = x'wx''$ for some $x', x'' \in V_G^*$,
– any letter of R is contained in x, and no letter of Q occurs in x.
The language L(G) generated by G is defined as

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

where $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$.

A *extended tabled interactionless L system* (ET0L system) is an $(r+3)$-tuple $G = (V, T, P_1, P_2, \ldots P_r, w)$ where
– V is an alphabet, T is a subset of V,
– w is a non-empty word over V and,
– for $1 \le i \le r$, P_i is a finite subset of $V \times V^*$ such that, for any $a \in V$, there is at least one element (a, v) in P_i.
Again, we shall write $a \to v$ instead of (a, v). $x \in V^+$ directly derives $y \in V^*$ (written as $x \Longrightarrow y$), if
– $x = x_1x_2 \ldots x_n$ for some $n \ge 0$, $x_i \in V$, $1 \le i \le n$,
– $y = y_1y_2 \ldots y_n$ and
– there is an j, $1 \le j \le r$ such that $x_i \to y_i \in P_j$ for $1 \le i \le n$.
The language $L(G)$ generated by the ET0L system G is defined as

$$L(G) = \{z \mid z \in T^*,\ w \Longrightarrow^* z\},$$

where $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$.

A *random context ET0L system* (RCET0L system in short) is an $(r+3)$-tuple $G = (V, T, P_1, P_2, \ldots P_r, w)$ where
– for $1 \le i \le r$, $P_i = (P_i', R_i, Q_i)$ where R_i and Q_i are subsets of V,
– $G' = (V, T, P_1', P_2', \ldots P_r', w)$ is an ET0L system.
$x \in V^+$ directly derives $y \in V^*$ (written as $x \Longrightarrow y$), if
– any letter of R_i occurs in x and no letter of Q_i occurs in x, and
– $x \Longrightarrow y$ holds with respect P_i' in G'.
The language $L(G)$ generated by the RCET0L system G is defined as

$$L(G) = \{z \mid z \in T^*,\ w \Longrightarrow^* z\},$$

where $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$.

By $\mathcal{L}(CF)$, $\mathcal{L}(RC)$, $\mathcal{L}(fRC)$, $\mathcal{L}(rp)$, Ł$(ET0L)$ and $\mathcal{L}(RCET0L)$ we denote the families of all context-free languages, random context languages, forbidden random context languages, Russian parallel languages, ET0L languages and random context

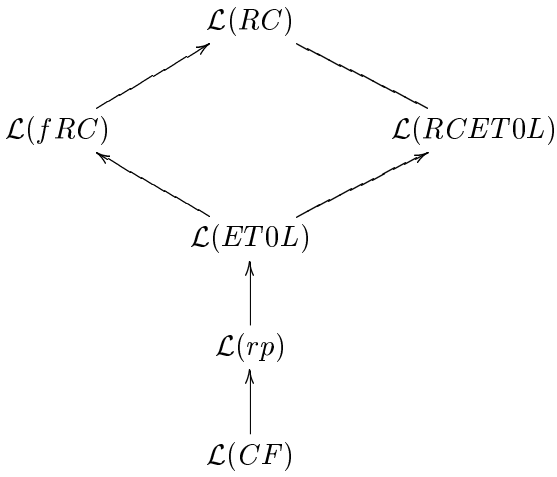

Figure 1:

ET0L languages. For a detailed information on these languages we refer to [10], [12], [17] and [16].

In Figure 1 we recall the hierarchy of the languages defined above.

3 Cooperating Distributed Grammar Systems

We now present the notion of cooperating distributed grammar systems.

A *cooperating distributed grammar system* (for short, CD grammar system is an $(n+5)$-tuple

$$G = (N, T, P_1, P_2, \ldots, P_n, S, c, c'),$$

where
– N is a set of nonterminals and T is a set of terminals,
– for $1 \leq i \leq n$, the component P_i is a set of context-free productions,
– $S \in N$ is the start element,
– c is the start condition and c' is the stop condition, i.e., c and c' are predicates on

$$\{(P, w) \mid P \subset N \times (N \cup T)^* \text{ and } w \in (N \cup T)^*\}$$

(the conditions are defined on pairs consisting of a set of context-free productions and a word).

We say that

$$x = x_0 \Longrightarrow x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_m = y$$

is a derivation with respect to P_i and the conditions c and c' (written as $x \Longrightarrow^*_{P_i} y$) if
– $c(P_i, x)$ is true,
– for $1 \leq j \leq m+1$, $x_j \Longrightarrow x_{j+1}$ is a direct derivation step using a production of P_i
– for $1 \leq j \leq m-1$, $c'(P_i, x_j)$ is not true, and
– $c'(P_i, y)$ is true or y contains no letter of $dom(P_i)$.

The language generated by G consists of all words z over T such that there is derivation

$$S = z_0 \Longrightarrow^*_{P_{i_1}} z_1 \Longrightarrow^*_{i_2} z_2 \Longrightarrow^*_{P_{i_3}} \ldots \Longrightarrow^*_{P_{i_m}} z_m = z\,.$$

In this paper we discuss CD grammar systems where the start and stop conditions depend on the competence of the components to work on the sentential form. The competence is formalized in the following way.

Let N and T be two disjoint sets (of nonterminals and terminals, respectively). Further, let $V = N \cup T$ and let $P \subseteq N \times V^*$ be a (finite) set of context-free productions and w be a word over V. We set

$$nt(w) = \{A \mid A \in N \text{ and } \#_A(w) \geq 1\}$$

(i.e., $nt(w)$ is the set of nonterminals which occur in w),

$$dom(P) = \{A \mid A \to w \in P \text{ for some } w \in V^*\}$$

and

$$comp(P_i, w)) = \#(nt(w) \cap dom(P_i))\,.$$

We say that P has

competence k on w	iff	$comp(P_i, w) = k$,
competence $\leq k$ on w	iff	$comp(P_i, w) \leq k$,
competence $\geq k$ on w	iff	$comp(P_i, w) \geq k$,
maximal competence on w	iff	$comp(P_i, w) \geq comp(P_j, w)$ for $1 \leq j \leq n$,
full competence on w	iff	$nt(w) \subseteq dom(P_i)$.

The most obvious idea is to require that a component can start a derivation if it has a certain competence A and has to stop if it does not have the competence A. Therefore besides the competence conditions introduced above we have to consider the negations of the above conditions. Clearly, the negations can also be used as start conditions. Obviously, P has not competence $\geq k$ ($\leq k$) iff it has competence $\leq k-1$ ($\geq k+1$, respectively). Therefore we get the following additional notions.

P_i has competence $\neq k$ on w	iff	$comp(P_i, w) \neq k$,
P_i is not maximal competent on w	iff	there is a P_j, $1 \leq j \leq n$, $i \neq j$, with $comp(P_j, w) > comp(P_i, w)$,
P_i is not fully competent on w	iff	there is an $A \in nt(w)$ with $A \notin dom(P_i)$.

By these definitions, competence of P_i on w is given, if $dom(P_i)$ contains at least one element of $nt(w)$. Thus we also formulate:

P_i is competent on w	iff	$comp(P_i, w) \geq 1$,
P_i is not competent on w	iff	$comp(P_i, w) = 0$.

We use the following abbreviations for the competence conditions (in the order of their definition):

$$= k,\ \leq k,\ \geq k,\ max,\ full,\ \neq k,\ \neg max,\ \neg full,\ comp,\ \neg comp\,.$$

Because *comp* and ≥ 1, $\neq 1$ and ≥ 2 as well as ≤ 1 and $= 1$ coincide, we can restrict to $k \geq 2$ and have to add the condition $= 1$.

As an example we consider the CD grammar system

$$\begin{aligned}
G &= (\{S, A, B, A', B', F\}, \{a, b, c\}, P_1, P_2, P_3, P_4, P_5, P_6, S, max, \neg max), \\
P_1 &= \{S \to AB\}, \\
P_2 &= \{A \to ab, B \to c\}, \\
P_3 &= \{A \to aA'b, B \to F\}, \\
P_4 &= \{A' \to F, B \to B'c\}, \\
P_5 &= \{A' \to A, B' \to F\}, \\
P_6 &= \{A \to F, B' \to B\}.
\end{aligned}$$

Obviously, after the application of P_1, P_2 and P_3 have maximal competence. If we apply P_2, then we have to terminate the derivation and get abc. If we apply P_3, we only can apply $A \to aA'b$ and loose the maximal competence. We have to apply in succession $B \to B'c$ of P_4, $A' \to A$ of P_5 and $B' \to B$ of P_6 and obtain $aAbBc$ (if we apply the other rule of the components we get a sentential form containing F and we cannot terminate the derivation). We can iterate this process. Therefore

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}.$$

Note that we cannot combine an arbitrary competence condition as start condition with another arbitrary competence condition as a stop condition. Obviously, we cannot take the same condition as start condition as well as stop condition, because by definition the component can start the derivation but it also has to stop the derivation, which is impossible. However, there are also other situations of this type. For instance, $full$ and max cannot be taken as start and stop condition, respectively, since any full competent component is maximal, too.

Moreover, we have to exclude those combination which do not allow terminating derivations. For instance, this situation occurs in the case of start condition ≥ 4 and stop condition ≤ 2. If a derivation starts then the sentential word has at least four nonterminals. However, if the derived sentential form contains at most 2 nonterminals, we have to stop the derivation (and no component can continue the derivation by the start condition). Therefore we cannot derive terminal words. This situation also occurs if we have stop conditions of the types $\leq k$ with $k \geq 2$, $= 1$ and *comp*.

Furthermore, we shall exclude such situations where the start condition as well as the stop condition can be satisfied, but it is not necessarily true that both have to be valid. For instance, if $N = \{A, B, C\}$ and the start condition requires the competence ≥ 2 and the stop condition requires competence ≥ 3, then the derivation can start start on AB, but it cannot start on ABC since the start condition as well as the stop condition are satisfied. The pairs $(max, full)$ and $(full, = k)$ with $k \geq 2$ of start and stop conditions can also lead to conflicts.

Taking all these restrictions into consideration we have to investigate the following pairs of of start conditions and stop conditions, where $k \geq 2$ and $l \geq 2$:

(a) $(full, \neg full)$, $(full, \neg comp)$, $(max, \neg max)$, $(max, \neg comp)$, $(comp, \neg comp)$, $(= l, \neg comp)$, $(\geq l, \neg comp)$, $(\leq l, \neg comp)$, $(= 1, \neg comp)$, $(= 1, \geq k)$, $(= 1, = k)$,

(b) $(full, \neg max)$, $(\neg full, full)$, $(\neg full, \neg comp)$, $(\neg max, max)$, $(\leq l, = k)$ and $(\leq l, \geq k)$ with $l < k$, $(= l, = k)$ with $l \neq k$, $(= l, \geq k)$ with $l < k$, $(\neq k, = k)$

By $\mathcal{L}(c, c')$ we denote the family of all languages which can be generated by CD grammar systems with the start condition c and the stop condition c'. If we restrict the CD grammar systems which have at most n components, then we denote the corresponding family of languages by $\mathcal{L}_n(c, c')$.

By definition, we have the following statement.

Lemma 1 *For any $n \geq 1$ and any pair (c, c') of start and stop conditions,*

$$\mathcal{L}_n(c, c') \subseteq \mathcal{L}_{n+1}(c, c') \subseteq \mathcal{L}(c, c')\,.$$

4 Results

In this section we present the results which – at least partially – give the place of some families $\mathcal{L}(c, c')$ within the hierarchy given in Figure 1.

Theorem 2 *([6]) For any $n \geq 3$,*

$$\begin{aligned} \mathcal{L}(CF) &= \mathcal{L}_1(comp, \neg comp) = \mathcal{L}_2(comp, \neg comp) \\ &\subset \mathcal{L}_n(comp, \neg comp) = \mathcal{L}(comp, \neg comp) = \mathcal{L}(ET0L)\,. \end{aligned}$$

Theorem 3 *([14], [5], [4]) For any $n \geq 3$,*

$$\begin{aligned} \mathcal{L}(CF) &= \mathcal{L}_1(full, \neg full) \subset \mathcal{L}_2(full, \neg full) \\ &\subseteq \mathcal{L}_n(full, \neg full) = \mathcal{L}(full, \neg full) = \mathcal{L}(RC)\,. \end{aligned}$$

Theorem 4 *For any $n \geq 3$,*

$$\begin{aligned} \mathcal{L}(CF) &= \mathcal{L}_1(full, \neg comp) \subseteq \mathcal{L}_2(full, \neg comp) \\ &\subseteq \mathcal{L}_n(full, \neg comp) = \mathcal{L}(full, \neg comp) = \mathcal{L}(ET0L)\,. \end{aligned}$$

Proof. $\mathcal{L}(full, \neg comp) \subseteq \mathcal{L}(ET0L)$. Let $L \in \mathcal{L}(full, \neg comp)$. Then $L = L(G)$ for some CD grammar system $G = (N, T, P_1, P_2, \ldots, P_r, w, full, \neg comp)$. We set

$$\begin{aligned} N' &= N \cup \{A' \mid A \in N\} \cup \{F\}, \\ h_i(a) &= \begin{cases} a & \text{if } a \in dom(P_i) \cup T \\ a' & \text{if } a \notin dom(P_i) \end{cases} \quad \text{for } 1 \leq i \leq n, \\ Q_i &= \{A \to h_i(w) \mid A \to w \in P_i\} \cup \{B \to F \mid B \notin dom(P_i)\}, \text{ for } 1 \leq i \leq n, \\ Q &= \{A' \to A \mid A \in N\}, \\ G' &= (N', T, Q_1, Q_2, \ldots Q_r, Q, w, comp, \neg comp) \end{aligned}$$

Let $1 \leq i \leq n$. If a word x contains a symbol $B \notin dom(P_i)$, then Q_i derives in G' from x a word containing the letter F. Since there is no rule with left-hand side F in any component of G', we cannot terminate the derivation. Therefore in a terminating derivation we can only apply a component Q_i to x if $nt(x) \cap (N \setminus dom(P_i) = \emptyset$, or equivalently, $nt(x) \subseteq dom(P_i)$, i.e., P_i is fully competent on x. Therefore Q_i is applicable in a terminating derivation in G' to x if and only if P_i is fully competent on x if and only if P_i is applicable to x in G.

Moreover, in both cases we only stop if the component is not competent on the derived sentential form. Since we introduce primed versions of the letters of $N \setminus dom(P_i)$ applying Q_i we obtain

$$x \Longrightarrow_{P_i}^* y \quad \text{iff} \quad x \Longrightarrow_{Q_i}^* h_i(y)\,.$$

Furthermore, no component Q_i with $1 \leq i \leq n$ is competent on $h_i(y)$ since it only contains primed letters and terminals. Thus we have to continue with Q which replaces all primed letters A' by their original A. Thus

$$x \Longrightarrow_{P_i}^* y \quad \text{iff} \quad x \Longrightarrow_{Q_i}^* h_i(y) \Longrightarrow_{Q}^* y\,.$$

This implies $L(G) = L(G')$. By Lemma 2, $L = L(G) = L(G') \in \mathcal{L}(ET0L)$.

$\mathcal{L}(ET0L) \subseteq \mathcal{L}_3(full, \neg comp)$. In [6], Theorem 3, iii), it has been shown that, for any ET0L language L, there is a cooperating distributed grammar system $G = (N, T, P_1, P_2, P_3, S, comp, \neg comp)$ such that $L = L(G)$. Moreover, it is easy to see, that any component of G is competent on a sentential form x if and only if it is fully competent on x. Hence $G' = (N, T, P_1, P_2, P_3, S, full, \neg comp)$ generates L, too. Thus $\mathcal{L}(ET0L) \subseteq \mathcal{L}_3(full, \neg comp)$.

By Lemma 1, we obtain

$$\mathcal{L}(ET0L) \subseteq \mathcal{L}_3(full, \neg comp) \subseteq \mathcal{L}_n(full, \neg comp) \subseteq \mathcal{L}(full, \neg comp) \subseteq \mathcal{L}(ET0L)$$

for any $n \geq 4$, which implies

$$\mathcal{L}_m(full, \neg comp) = \mathcal{L}(full, \neg comp) = \mathcal{L}(ET0L)$$

for any $m \geq 3$. Obviously, $\mathcal{L}(CF) = \mathcal{L}_1(full, \neg comp)$. Now the assertion follows by Lemma 1. □

Theorem 5 *([7])* $\mathcal{L}(rp) \subset \mathcal{L}(max, \neg max) \subseteq \mathcal{L}(RC)$.

Theorem 6 $\mathcal{L}(max, \neg comp) = \mathcal{L}(RCET0L)$.

Proof. $\mathcal{L}(RCET0L) \subseteq \mathcal{L}(max, \neg comp)$. Let $L \in \mathcal{L}(RCET0L)$. Then there is a random context ET0L system

$$G = (V, T, (P_1, R_1, Q_1), (P_2, R_2, Q_2), \ldots, (P_r, R_r, Q_r), w)$$

such that $L = L(G)$. Let

$$\begin{aligned} V &= \{a_1, a_2, \ldots, a_n\}\,, \\ R_i &= \{a_{i,1}, a_{i,2}, \ldots, a_{i,s_i}\} \text{ for } 1 \leq i \leq r\,. \end{aligned}$$

We set

$$
\begin{aligned}
N &= \{a' \mid a \in V\} \cup \{a'' \mid a \in V\} \cup \{A, S, F\} \cup \{B_i \mid 1 \le i \le n+1\} \\
&\quad \cup \{A_{i,j,k} \mid 1 \le i \le r, 1 \le j \le s_i + 2, 1 \le k \le n+1\}, \\
Z_{init} &= \{S \to AA_{i,1,1}A_{i,1,2} \dots A_{i,1,n+1}w' \mid 1 \le i \le r\} \\
&\quad \cup \{S \to B_1B_2 \dots B_{n+1}w'\}, \\
Z_{fin} &= \{B_i \to \lambda \mid 1 \le i \le n+1\} \cup \{a' \to F \mid a' \in N \setminus T'\} \cup \{a' \to a \mid a \in T\}, \\
Z_{i,j} &= \{A_{i,j,k} \to A_{i,j+1,k} \mid 1 \le k \le n+1\} \cup \{a'_{i,j} \to a''_{i,j}\} \\
&\quad \text{for } 1 \le i \le r, 1 \le j \le s_i, \\
Z'_{i,j} &= \{A_{i,j,k} \to A_{i,j+1,k} \mid 1 \le k \le n+1\} \cup \{A \to F\} \\
&\quad \text{for } 1 \le i \le r, 1 \le j \le s_i, \\
Z_{i,s_i+1} &= \{A_{i,s_i+1,k} \to A_{i,s_i+2,k} \mid 1 \le k \le n+1\} \cup \{a' \to a'' \mid a \in V \setminus Q_i\} \\
&\quad \cup \{b' \to F \mid b \in Q_i\} \text{ for } 1 \le i \le r, \\
Z_{i,s_i+2} &= \{A_{i,s_i+2,k} \to \lambda \mid 1 \le k \le n+1\} \cup \{a'' \to u' \mid a \to u \in P_i\} \\
&\quad \cup \{A \to AA_{i',1,1}A_{i',1,2} \dots A_{i',1,n+1} \mid 1 \le i' \le r\} \\
&\quad \cup \{A \to B_1B_2 \dots B_{n+1}\} \text{ for } 1 \le i \le r, \\
G' &= (N, T, Z_{init}, Z_{fin}, Z_{1,1}, Z_{1,2} \dots, Z_{1,s_1+2}, Z_{2,1}, \dots, Z_{r,s_r+2}, \\
&\quad Z'_{1,1}, Z'_{1,2} \dots, Z'_{1,s_1}, Z'_{2,1}, \dots, Z'_{r,s_r}, S, max, \neg comp).
\end{aligned}
$$

If we start a derivation in G' (from S) we get $AA_{i,1,1}A_{i,1,2} \dots A_{i,1,n+1}w'$ for some i, $1 \le i \le r$, or $B_1B_2 \dots B_{n+1}w'$.

We now discuss the continuation of the derivation of a sentential form of type $B_1B_2 \dots B_{n+1}v'$ where v is a sentential form of G. Since v' contains at most n different letters, Z_{fin} is the only component with maximal competence and we derive the terminal word v or a word containing an occurrence of F, i.e., the derivation cannot be terminated. If v is a terminal word, then $v \in L(G)$ as well as $v \in L(G')$.

Now let us consider the derivation starting from $AA_{i,1,1}A_{i,1,2} \dots A_{i,1,n+1}v'$ for some i, $1 \le i \le r$ and some sentential form of G. Now $Z'_{i,1}$ has maximal competence, and its applications yields a word containing F such that we cannot terminate the derivation. If $a'_{i,1}$ is present, then $Z_{i,1}$ has also maximal competence, and its application yields $AA_{i,2,1}A_{i,2,2} \dots A_{i,2,n+1}v_1$ where v_1 is obtained from v' by replacing any occurrence of $a'_{i,1}$ by $a''_{i,1}$. Therefore we cannot terminate the derivation or all letters of R_i are present in v and we have a derivation

$$AA_{i,1,1}A_{i,1,2} \dots A_{i,1,n+1}v' \Longrightarrow^*_{Z_{i,1}} \dots \Longrightarrow^*_{Z_{i,r_i}} AA_{i,r_i+1,1}A_{i,r_i,2} \dots A_{i,r_i,n+1}v_2$$

where all occurrences of primed versions of letters of R_i in v' are replaced by their doubly primed versions to obtain v_2. Now Z_{i,r_i+1} is the only component with maximal competence, and its application yields a word containing F, if v' contains a primed version of a letter in Q_i, or we get $AA_{i,r_i+2,1}A_{i,r_i+2,2} \dots A_{i,r_i+2,n+1}v''$. Now we have to apply the only component Z_{i,r_i+2} with maximal competence which results in $AA_{i',1,1}A_{i',1,2} \dots A_{i',1,n+1}z'$ or $B_1B_2 \dots B_{n+1}z'$ where $v \Longrightarrow_{P_i} z$ is a derivation step in the RCET0L system G.

Hence we can simulate in G' all derivations in G, and any terminating derivation In G' corresponds to a derivation in G. Therefore $L(G') = L(G) = L$ which proves $L \in \mathcal{L}(max, \neg comp)$.

$\mathcal{L}(max, \neg comp) \subseteq \mathcal{L}(RCET0L)$. Let $L \in \mathcal{L}(max, \neg comp)$. Then $L = L(H)$ for some CD grammar system $H = (N, T, P_1, P_2, \ldots, P_n, S, max, \neg comp)$. Obviously, for any word $w \in (N \cup T)^*$, the set $nt(w)$ uniquely determines the set of components which are of maximal competence on w. Let $f : 2^N \to 2^{\{P_1, P_2, \ldots, P_n\}}$ be the function where $f(M)$ is the set of components which have maximal competence on words w with $M = nt(w)$.

We now construct the random context ET0L system H' with the the underlying alphabet $V = N \cup T \cup \{A\} \cup \{[i] \mid 1 \leq i \leq n\} \cup \{A_M \mid M \subseteq N\}$, the terminal alphabet T, the start word AS and the following tables:

$$(\{A \to A_M\} \cup \{x \to x \mid x \in V \setminus \{A\}\}, \{A\} \cup M, N \setminus M) \text{ for } M \subseteq N$$

(this table is only applicable to a word Av with $M = nt(v)$, i.e., by this table we determine $nt(v)$, and we obtain $A_{nt(v)}v$),

$$(A_M \to [i] \mid P_i \in f(M)\} \cup \{x \to x \mid x \in V \setminus \{A_M\}\}, \{A_M\}, \emptyset) \text{ for } M \subseteq N$$

(by this table we derive $[i]v$ from $A_{nt(v)}v$, where P_i has maximal competence on v),

$$\begin{aligned}&([i] \to [i]\} \cup P_i \cup \{x \to x \mid x \in V \setminus (dom(P_i) \cup \{[i]\}\}, \{[i]\}, \emptyset)\,,\\ &([i] \to A\} \cup \{x \to x \mid x \in V \setminus \{[i]\}\}, \{[i]\}, dom(P_i))\end{aligned}$$

(by these tables, in H' we obtain a derivation

$$[i]v \Longrightarrow [i]v_1 \Longrightarrow [i]v_2 \Longrightarrow \ldots \Longrightarrow [i]v_m \Longrightarrow Az$$

where $v \Longrightarrow^*_{P_i} z$ holds in G, P_i has maximal competence on v and P_i is not competent on z),

$$(\{A \to \lambda\} \cup \{x \to x \mid x \in N \setminus \{A\}, \{A\}, N)$$

(we cancel A and produce a word over T). By the explanations added to the tables, it is easy to see that $L(H') = L(H) = L$ which proves $L \in \mathcal{L}(RCET0L)$. □

Theorem 7 *([1])* $\mathcal{L}(fRC) \subseteq \mathcal{L}(= 1, \geq 2) \subseteq \mathcal{L}(RC)$.

Theorem 8 *([8])* $\mathcal{L}(= 1, \neg comp) = \mathcal{L}(\geq 1, \neg comp) = \mathcal{L}(ET0L)$.

Theorem 9 *([8]) For all* $k \geq 2$, $\mathcal{L}(= k, \neg comp) = \mathcal{L}(\geq k, \neg comp) = \mathcal{L}(RCET0L)$.

Theorem 10 *([8]) For all* $k \geq 1$, $\mathcal{L}(\leq k, \neg comp) = \mathcal{L}(ET0L)$.

5 Concluding Remarks

In the preceding section we have characterized – in some cases only partially – the families given in item (a) in the list at the end of Section 3. We do not know any result on the families mentioned in item (b) such that the generative power of these families has to be studied.

Above we have mentioned that some combinations of start and stop conditions do not allow the derivation of terminal words. Especially, this holds for the combinations $(= k, \neq k)$ and $(\geq k, \leq k-1)$ for any $k \geq 2$. In order to get terminating derivations in these cases one has some possibilities.

In [1] and [2] the authors allow that there is a special terminating phase. If this phase is started, then the component has to stop with a terminal word and for all intermediate sentential forms of this step the competence of component is at most k.

Another approach is presented in [3] where in each derivation of a component one has to perform a parallel derivation as in L systems.

For the classical types of stop conditions hybrid systems have been introduced, i.e., the start and stop condition is not associated with the system, it is only associated with a component, and the start conditions and/or the stop conditions can be different for different components (see e.g. [15] and [13]). Such hybrid systems can also be defined if one uses competence based start and stop conditions.

References

[1] M. ter Beek, E. Csuhaj-Varjú, M. Holzer and Gy. Vaszil, On competence in cooperating distributed grammar systems. Submitted, 2004.

[2] M. ter Beek, E. Csuhaj-Varjú, M. Holzer and Gy. Vaszil, On competence in cooperating distributed grammar systems, part II. Submitted, 2004.

[3] M. ter Beek, E. Csuhaj-Varjú, M. Holzer and Gy. Vaszil, On competence in cooperating distributed grammar systems with parallel rewriting. Submitted, 2004.

[4] H. Bordihn, On the number of components in CD grammar systems. In: E. Csuhaj-Varjú and Gy. Vaszil (Eds.), Proceedings of the Conference on Descriptional Complexity of Formal Systems, Budapest, 2003.

[5] H. Bordihn and E. Csuhaj-Varjú, On competence and completeness in CD grammar systems. *Acta Cybernetica* **12** (1996) 347–361.

[6] E. Csuhaj-Varjú and J. Dassow, On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics* (EIK) **26** (1990), 49–63.

[7] E. Csuhaj-Varjú, J. Dassow and M. Holzer, On a competence-based cooperation strategy in CD grammar systems. Submitted, 2004.

[8] E. Csuhaj-Varjú, J. Dassow and M. Holzer, CD grammar systems with competence based entry conditions in their cooperation protocols. Submitted, 2004.

[9] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun, *Grammar Systems - A Grammatical Approach to Distribution and Cooperation.* Topics in Computer Mathematics 5, Gordon and Breach Science Publishers, Yverdon, 1994.

[10] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory.* EATCS Monograph on Theoretical Computer Science 18, Springer-Verlag, Berlin-Heidelberg-New York, 1989.

[11] J. Dassow, Gh. Păun and G. Rozenberg, Grammar systems. In: [17], Vol. II, Chapter 4, 155-213.

[12] J. Dassow, Gh. Păun and A. Salomaa, Grammar with controlled derivations. In: [17], Vol. II, Chapter 3, 101-154.

[13] H. Fernau, M. Holzer and R. Freund, Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theor. Comp. Sci.* **259** (2001) 404–426.

[14] R. Meersman and G. Rozenberg, Cooperating grammar systems. In: J. Winkowski (ed.), Proceeding Mathematical Foundations of Computer Science 1978, Lecture Notes in Computer Science 64, Springer-Verlag, Berlin, 1978, 364–374.

[15] V. Mitrana, Hybrid cooperating/distributed grammar systems. *Computer and artrificial Intelligence* **12** (1993) 83–88.

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

[17] G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages.* Vol. I – III, Springer Verlag, Berlin-Heidelberg-New York, 1997.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 170 - 178, 2004.

Simple Splicing Grammar Systems

K. S. Dersanambika*, K. Krithivasan

Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai - 600 036, India
kamala@iitm.ernet.in

K. G. Subramanian

Department of Mathematics, Madras Christian College
Chennai - 600 059, India
kgsmani1948@yahoo.com

Abstract

Dassow and Mitrana [2] introduced a new type of grammar system called splicing grammar system in which communication is done by splicing of strings. Thus segments of sentential forms determined by given splicing rules are exchanged. In this paper, we consider simple splicing rules in these grammar systems and thus obtain four types of simple splicing grammar systems (SSGS), namely, $< 1, 3 >$, $< 2, 4 >$, $< 1, 4 >$, $< 2, 3 >$ SSGS. As $< 1, 3 >$ and $< 2, 4 >$ types of SSGS are equivalent and $< 2, 3 >$ and $< 1, 4 >$ types of SSGS become equivalent, there are essentially two types. Various properties of simple splicing grammar systems are obtained by considering different component grammars. We prove that context free simple splicing grammar systems with two components can generate context sensitive languages. Moreover systems with two regular components can generate nonlinear and context-free languages.

1 Introduction

The theory of Grammar Systems [1] is an intensively investigated area of Formal Language Theory providing an effective grammatical framework for capturing several phenomena characteristic of multi-agent systems such as cooperation, distribution, communication, parallelism etc. The basic idea in a grammar system is to consider several usual grammars and to make them cooperate in order to generate a common language. In parallel communicating grammar systems, the components are generative grammars working on their own sentential forms in parallel and communicating by request.

Motivated by the behaviour of DNA sequences under the influence of restriction enzymes and ligases, Head [3, 4] defined splicing systems that make use of a new

*This work is partially supported by University Grants Commision, India

operation, called splicing on strings of symbols. The idea here is that given two strings, each of these is "cut" at suitable "sites" and "pasted crosswise" yielding new strings. Dassow and Mitrana [2] introduced a new type of parallel communicating grammar systems by replacing communication by splicing of strings, thus exchanging segments of sentential forms determined by given splicing rules. Paun [6] has investigated splicing grammar systems improving the results of [2].

Mateescu et al [5] have considered simple splicing systems that make use of splicing rules that are as simple as possible. In this paper we examine splicing grammar systems by requiring the splicing rules to be simple in the sense of [5]. Various properties of the resulting simple splicing grammar systems are obtained by considering different component grammars.

2 Preliminaries

For basic results of formal language theory one can refer to [7]. For notions and results pertaining to grammar system we refer to [1]. We denote the family of regular and context-free languages by REG and CF respectively.

For an alphabet V, the set of all words over V is denoted by $V^*$and the empty word by λ; moreover $V^+ = V^* - \{\lambda\}$. We recall the definition of a simple splicing system [5].

Definition 1 A simple splicing system is a triple

$$\Gamma = (V, A, M)$$

where V is an alphabet, $A \subseteq V^*$ is a finite set of axioms and $M \subseteq V$.

The elements of M are called markers. One can consider four types of languages over V^*, corresponding to the splicing rules of the forms

$$a\#\$a\#, \#a\$\#a, a\#\$\#a, \#a\$a\#$$

where a is an arbitrary element of M. These four rules are respectively called splicing rules of type $< 1, 3 >$, $< 2, 4 >$, $< 1, 4 >$, $< 2, 3 >$.

Clearly splicing rules of types $< 1, 3 >$ and $< 2, 4 >$ yield the same result and for $x, y, z \in V^*$ and $a \in M$ we obtain
$(x, y) \vdash^a_{<(1,3>} z$ iff $x = x_1ax_2, y = y_1ay_2, z = x_1ay_2$, for some $x_1, x_2, y_1, y_2 \in V^*$

For the other types, the splicing is performed as follows:
$(x, y) \vdash^a_{<1,4>} z$ iff $x = x_1ax_2, y = y_1ay_2, z = x_1aay_2$, for some $x_1, x_2, y_1, y_2 \in V^*$
$(x, y) \vdash^a_{<2,3>} z$ iff $x = x_1ax_2, y = y_1ay_2, z = x_1y_2$, for some $x_1, x_2, y_1, y_2 \in V^*$

We now define simple splicing grammar systems.

Definition 2 A $< 1, 3 >$-simple splicing grammar system ($< 1, 3 >$-SSGS)of degree n is a construct

$$\Gamma = (N, T, (S_1, P_1), (S_2, P_2), \cdots, (S_n, P_n), M)$$

where

(i) N, T are disjoint alphabets and P_i, $1 \leq i \leq n$ are finite sets of production rules over $N \cup T$.
(ii) M is a finite subset of $(N \cup T)\#\$(N \cup T)\#$ with $\#$, $\$$ two distinct symbols which are not in $N \cup T$. Each element of M is a $< 1, 3 >$-simple splicing rule.

The sets P_i are called the components of Γ. We can consider grammars of the form $G_i = (N, T, S_i, P_i)$, $1 \leq i \leq n$. By a configuration, we mean an n-tuple consisting of words over $N \cup T$.
For Two configurations,
$x = (x_1, x_2, \cdots, x_n)$, $x_i \in (N \cup T)^* N (N \cup T)^*$, $1 \leq i \leq n$
$y = (y_1, y_2, \cdots, y_n)$, $y_i \in (N \cup T)^*$, $1 \leq i \leq n$
we define $x \Rightarrow_\Gamma y$ if and only if any of the following two conditions holds:
(i)for each $1 \leq i \leq n$, $x_i \Rightarrow_{P_i} y_i$,
(ii) there exist $1 \leq i, j \leq n$ such that
$x_i = x_i^{'} a x_i^{''}$, $x_j = x_j^{'} a x_j^{''}$,
$y_i = x_i^{'} a x_j^{''}$, $y_j = x_j^{'} a x_i^{''}$, for $a\#\$a\# \quad \in M$, and
$y_k = x_k$, for $k \neq i, j$.

In the derivation $x \Rightarrow_\Gamma y$, in $< 1, 3 >$-SSGS, (i) defines a rewriting step, but (ii) defines a $< 1, 3 >$-splicing step, corresponding to a communication step in a parallel communicating grammar system. There is no priority of any of these operations over the other.

A $< 2, 3 >$-simple splicing grammar system is analogously defined.

Definition 3 A $< 2, 3 >$-simple splicing grammar system ($< 2, 3 >$-SSGS) of degree n is a construct

$$\Gamma = (N, T, (S_1, P_1), (S_2, P_2), \cdots, (S_n, P_n), M)$$

where
(i) N, T are disjoint alphabets and P_i, $1 \leq i \leq n$ are finite sets of production rules over $N \cup T$.
(ii) M is a finite subset of $\#(N \cup T)\$(N \cup T)\#$ with $\#$, $\$$ two distinct symbols which are not in $N \cup T$.Each element of M is a $< 2, 3 >$-simple splicing rule.

The sets P_i are called the components of Γ. We can consider grammars of the form $G_i = (N, T, S_i, P_i)$, $1 \leq i \leq n$. By a configuration, we mean an n-tuple consisting of words over $N \cup T$.
For Two configurations,
$x = (x_1, x_2, \cdots, x_n)$, $x_i \in (N \cup T)^* N (N \cup T)^*$, $1 \leq i \leq n$
$y = (y_1, y_2, \cdots, y_n)$, $y_i \in (N \cup T)^*$, $1 \leq i \leq n$
we define $x \Rightarrow_\Gamma y$ if and only if any of the following two conditions holds:
(i)for each $1 \leq i \leq n$, $x_i \Rightarrow_{P_i} y_i$,
(ii) there exist $1 \leq i, j \leq n$ such that
$x_i = x_i^{'} a x_i^{''}$, $x_j = x_j^{'} a x_j^{''}$,
$y_i = x_i^{'} x_j^{''}$, $y_j = x_j^{'} a a x_i^{''}$, for $a\#\$a\# \quad \in M$, and
$y_k = x_k$, for $k \neq i, j$.

In the derivation $x \Rightarrow_\Gamma y$, in $< 2,3 >$-SSGS, (i) defines a rewriting step, but (ii) defines a $< 2,3 >$-splicing step, corresponding to a communication step in a parallel communicating grammar system. Again there is no priority of any of these operations over the other.

Moreover at any instant only one splicing operation can take place in the $< 1,3 >$-SSGS and $< 2,3 >$-SSGS.

Also $< 1,3 >$-SSGS and $< 2,4 >$-SSGS are essentially the same. Likewise $< 2,3 >$-SSGS and $< 1,4 >$-SSGS are the same by definition.

The language generated by the ith component is defined by
$L_i(\Gamma) = \{x_i \in T^* | (S_1, S_2, \cdots, S_n) \Rightarrow^* (x_1, x_2, \cdots, x_n),\ x_j \in (N \cup T)^*,\ j \neq i\}$,
where $\Rightarrow^*$ is the reflexive and transitive closure of the relation $\Rightarrow$.

Two kinds of languages [2] can naturally be associated to a simple splicing grammar system. One of them is the language generated by a single component and, because no component is distinguished in any way , we may always choose the language generated by the first component. This language will be called the individual language of the system.

The second associated language will be the total language, namely

$$L_t(\Gamma) = \bigcup_{i=1}^{n} L_i(\Gamma)$$

Example 1 *Consider the $< 1,3 >$-SSGS with regular rewriting rules. Let*
$\Gamma_1 = (N, T, (S_1, P_1), (S_2, P_2), M)$,
$N = \{S_1, S_2, A, B\}$
$T = \{a, b, c\}$
$P_1 = \{S_1 \to aA,\ A \to aA,\ A \to c\}$
$P_2 = \{S_2 \to cB,\ B \to bB,\ B \to b\}$
$M = \{c\#\$c\#\}$
This system produces the languages
$L_1(\Gamma_1) = \{a^n cb^n | n \geq 1\} \cup \{a^n c | n \geq 1\}$
$L_2(\Gamma_1) = \{cb^n | n \geq 0\}$
Here the total language is
$L_t = \{a^n cb^n | n \geq 1\} \cup \{a^n c | n \geq 1\} \cup \{cb^n | n \geq 0\}$

Example 2 *Consider the $< 1,3 >$-SSGS with context free rewriting rules. Let*
$\Gamma_1 = (N, T, (S_1, P_1), (S_2, P_2), M)$,
$N = \{S_1, S_2, A, B\}$
$T = \{a, b, c, d\}$
$P_1 = \{S_1 \to aAbd,\ A \to aAb,\ A \to ab\}$
$P_2 = \{S_2 \to dcB,\ B \to cB,\ B \to c\}$
$M = \{d\#\$d\#\}$
This system produces the languages
$L_1(\Gamma_1) = \{a^n b^n dc^n | n \geq 1\} \cup \{a^n b^n d | n \geq 1\}$
$L_2(\Gamma_1) = \{dc^n | n \geq 0\}$
Here the total language is
$L_t = \{a^n b^n dc^n | n \geq 1\} \cup \{a^n b^n d | n \geq 1\} \cup \{dc^n | n \geq 0\}$

We denote the family of individual languages generated by simple splicing grammar systems of degree n, with components of type X by $IssgsL_n(X)$.
Similarly we denote the family of total languages generated by simple splicing grammar systems of degree n, with components of type X by $TssgsL_n(X)$.
where $X \in \{REG, CF\}$.

Remark (i) In Example 1 and Example 2, if we use the $< 2, 4 >$-splicing rules instead of the $< 1, 3 >$-splicing rules i.e. $M = \{\#c\$\#c\}$ in Example 1 and $M = \{\#d\$\#d\}$ in Example 2 , without changing the rewriting rules, then we obtain the same languages in Examples 1 and 2 .

(ii) In Example 1, if we use the $< 2, 3 >$-splicing rule $\#c\$c\#$, instead of the $< 1, 3 >$-splicing rule , without changing the rewriting rules, then we obtain the following languages:
$L_1(\Gamma_1) = \{a^n c^2 b^n | n \geq 1\} \cup \{a^n b^n | n \geq 1\} \cup \{a^n c | n \geq 1\}$
$L_2(\Gamma_1) = \{cb^n | n \geq 1\} \cup \{c^2\} \cup \{\varepsilon\}$
$L_t(\Gamma_1) = \{a^n c^2 b^n | n \geq 1\} \cup \{a^n b^n | n \geq 1\} \cup \{a^n c | n \geq 1\} \cup \{cb^n | n \geq 1\} \cup \{c^2\} \cup \{\varepsilon\}$

In Example 2, if we use the $< 2, 3 >$-splicing rule $\#c\$c\#$, instead of the $< 1, 3 >$-splicing rule , without changing the rewriting rules, then we obtain the following languages.
$L_1(\Gamma_2) = \{a^n b^n d^2 c^n | n \geq 1\} \cup \{a^n b^n c^n | n \geq 1\} \cup \{a^n b^n d | n \geq 1\}$
$L_2(\Gamma_2) = \{dc^n | n \geq 1\} \cup \{d^2\} \cup \{\varepsilon\}$
$L_t(\Gamma_1) = \{a^n b^n d^2 c^n | n \geq 1\} \cup \{a^n b^n c^n | n \geq 1\} \cup \{a^n b^n d | n \geq 1\}$
$\cup \{dc^n | n \geq 1\} \cup \{d^2\} \cup \{\varepsilon\}$

3 The Regular and CF Cases

Lemma 1 *Let Γ be a regular $< 2, 4 >$-SSGS of degree n, Then a regular $< 2, 4 >$-SSGS of degree n, Γ', exists such that $L_t(\Gamma') = L_1(\Gamma)$*

Proof. Let

$$\Gamma = (N, T, (S_1, P_1), (S_2, P_2), \cdots, (S_n, P_n), M),$$

be a splicing grammar with the above mentioned property. For proving the assertion, we construct the system

$$\Gamma = (N', T, (S_1, P_1), (S_2, P_2'), \cdots (S_n, P_n'), M),$$

where

$$N' = N \cup \{X\}$$

$$P_i' = \{A \to aB \mid A \to aB \in P_i\} \cup \{A \to aX \mid A \to a \in P_i\}, 2 \leq i \leq n.$$

By this construction, all languages $L_i(\Gamma'), 2 \leq i \leq n$, become empty and $L_1(\Gamma') = L_1(\Gamma)$ or, in other words, $L_t(\Gamma') = L_1(\Gamma)$. □

The above result holds for $< 1, 3 >$-SSGS systems also because these two systems are equivalent.

Theorem 1 *For $Y \in \{I, T\}$,*

1. *For* $X \in \{REG, CF\}, X = Y_{<1,3>}ssgsL_1(X)$
2. $REG \subset Y_{<1,3>}ssgsL_2(REG)$
3. $CF \subset Y_{<1,3>}ssgsL_2(CF)$
4. $Y_{<1,3>}ssgsL_2(REG)$ *contains nonlinear languages.*

Proof. The first statement immediately follows from definitions. The second statement is a consequence of example 1. The third statement is a consequence of example 2.

As far as the last statement is concerned, the individual language generated by the following $< 1, 3 >$-splicing grammar system is the non-linear language
$L = \{a^n cb^n a^m db^m | n, m \geq 1\}$:

$$\Gamma = (\{S_1, S_2, A, B, X, Y, Z\}, \{a, b, c, d\}, (S_1, P_1), (S_1, P_1), M)$$

$P_1 = \{S_1 \to aA, A \to aA, A \to cX, B \to aE, E \to aE, E \to dZ\}$
$P_2 = \{S_2 \to cB, B \to bB, X \to dY, Y \to bY, Y \to b\}$
$M = \{c\#\$c\#, d\#\$d\#\}$
A derivation in Γ runs as follows:

$$\begin{aligned}(S_1, S_2) &\Rightarrow^+ (a^n c|X, c|b^n B)\\ &\Rightarrow (a^n cb^n B, cX)\\ &\Rightarrow (a^n cb^n aE, cdY)\\ &\Rightarrow (a^n cb^n a^m E, cdb^{m-1}Y)\\ &\Rightarrow (a^n cb^n a^m d|Z, cd|b^m)\\ &\Rightarrow (a^n cb^n a^m db^m, cdZ),\end{aligned}$$

for some n and $m \geq 1$. Hence we get
$L_1(\Gamma) = L_t(\Gamma) = \{a^n cb^n a^m db^m \mid n, m \geq 1\}$ and $L_2(\Gamma) = \phi$ □

Theorem 2 $Y_{<2,3>}ssgsL_2(REG)$ *also contains nonlinear languages.*

Proof. In the proof of the fourth statement in Theorem 1, a $< 1, 3 >$-SSGS splicing grammar system was constructed which generated a nonlinear language. In this, if we use $< 2, 3 >$-splicing rules instead of $< 1, 3 >$-splicing rules, i.e if we change the rules in M as
$M = \{\#c\$c\#, \#d\$d\#\}$
we obtain
$L_1(\Gamma) = L_t(\Gamma) = \{a^n b^n a^m b^m \mid n, m \geq 1\} \cup \{a^n b^n a^m d^2 b^m | \, n, m \geq 1\} \cup \{a^n c^2 b^n a^m d^2 b^m | \, n, m \geq 1\} \cup \{a^n c^2 b^n a^m b^m | \, n, m \geq 1\}$ and $L_2(\Gamma) = \phi$. This proves the result. □

We show that there is an infinite hierarchy of the classes $Y_{<2,3>}ssgsL_n(CF)$for increasing n.

Theorem 3
$CF = Y_{<2,3>}ssgsL_1(CF) \subset Y_{<2,3>}ssgsL_2(CF) \subset \cdots \subset Y_{<2,3>}ssgsL_n(CF) \subset \cdots$
where Y *denotes the total language or the individual language generated in the system*

Proof. We consider the following $Y_{<2,3>}ssgs$ simple splicing grammar system

$$\Gamma = (\{S_1, \cdots, S_n, A_1, \cdots, A_n, D_1, \cdots, D_n\}, \{a, b\}, (S_1, P_1), (S_2, P_2), \cdots, (S_n, P_n), M)$$

with
$P_1 = \{S_1 \to aA_1bD_1, A_1 \to aA_1b, A_1 \to ab\}$

$P_2 = \{S_2 \to D_1aA_2bD_2, A_2 \to aA_2b, A_2 \to ab\}$
$\cdots$
$\cdots$
$\cdots$
$P_n = \{S_n \to D_{n-1}aA_nb, A_n \to aA_nb, A_n \to ab\}$

$M = \{\#D_i\$D_i\# : i = 1, 2, \cdots, n+1\}$
A useful sample derivation is as follows:
$(S_1, S_2, \cdots, S_n) \Rightarrow (aA_1bD_1, D_1aA_2bD_2, \cdots, D_{n-1}aA_nb)$

$\Rightarrow^+ (a^{p-1}A_1b^{p-1}D_1, D_1a^{p-1}A_2b^{p-1}D_2, \cdots, D_{n-1}a^{p-1}A_nb^{p-1})$

$\Rightarrow (a^pb^p|D_1, D_1|a^pb^pD_2, \cdots, D_{n-1}a^pb^p)$

$\Rightarrow (a^pb^pa^pb^p|D_2, , D_2|a^pb^pD_3 \cdots, D_{n-1}a^pb^p)$

$\cdots$
$\cdots$
$\cdots$
$\Rightarrow (a^pb^pa^pb^p \cdots a^pb^p, D_1^2, D_2^2, \cdots, D_{n-1}^2)$

Therefore we obtain $I_{<2,3>}ssgsL_n(CF) = T_{<2,3>}ssgsL_n(CF) = \{(a^pb^p)^n|\ p \geq 2\} \cup \{\epsilon\}$.

We can see that $L(\Gamma)$ is not in $Y_{<2,3>}ssgsL_{n-1}(CF)$. In fact in any component we can generate with CF rules only strings of the form a^nb^n prefixed with a nonterminal or suffixed with a nonterminal or both but not strings of the form $a^nb^nc^n$. When we splice strings from two components we get strings whose structure is similar to $a^nb^nc^nd^n$ but we cannot get strings with structure similar to $a^nb^nc^nd^ne^nf^n$ with two components. Continuing this argument, we find then $L(\Gamma)$ cannot be generated by a $Y_{<2,3>}ssgs$ language with $n-1$ components. □

Theorem 4 $Y_{<2,3>}ssgsL_2(REG) - CF \neq \phi$

Proof. Consider the following $< 2, 3 >$-simple splicing grammar system

$$\Gamma = (\{S_1, S_2, A, B, C, D\}, \{a, b, c, d\}, (S_1, P_1), (S_2, P_2), M)$$

$P_1 = \{S_1 \to aS_1, S_1 \to eA, X \to cX, X \to dD, C \to c\}$
$P_2 = \{S_2 \to dB, B \to bB, B \to eX, A \to cA, A \to cC, D \to dD\}$
$M = \{\#e\$e\#, \#d\$d\#\}$

A derivation in Γ runs as follows.

$$\begin{aligned}(S_1, S_2) &\to (aS_1, dB)\\ &\Rightarrow^+ (a^{n+1}S_1, db^nB)\\ &\Rightarrow (a^{n+1}|eA, db^ne|X)\\ &\Rightarrow (a^{n+1}X, db^ne^2A)\\ &\Rightarrow^+ (a^{n+1}c^mX, db^ne^2c^mA)\\ &\Rightarrow (a^{n+1}c^m|dD, d|b^ne^2c^{m+1}C)\\ &\Rightarrow (a^{n+1}c^mb^ne^2c^{m+1}C, d^2D)\\ &\Rightarrow (a^{n+1}c^mb^ne^2c^{m+2}, d^3D)\end{aligned}$$

Clearly, this language is Context-sensitive but not CF. This proves the result. □

In [2], in the conclusion, the question of whether the hierarchy $YsgsL_n(REG) \subseteq YsgsL_{n+1}(REG)$,$Y \in \{I, T\}$, is infinite, is left open. But the splicing operation is not restricted to be simple. Here we prove that the hierarchy is indeed infinite.

Theorem 5 $REG = YsgsL_1(REG) \subset YsgsL_2(REG) \subset \cdots \subset YsgsL_n(REG) \subset \cdots$

Proof. We consider the following splicing grammar system. Let

$$\Gamma = (\{S_1, \cdots, S_n, A_1, \cdots, A_n\}, \{a, b\}, (S_1, P_1), (S_2, P_2), \cdots, (S_n, P_n), M)$$

where for $i < n$ and i odd,
$P_i = \{S_i \to c_iA_i, A_i \to aA_i, A_i \to aD_{i+1}\}$
and for $i < n$ and i even,
$P_i = \{S_i \to c_iA_i, A_i \to bA_i, A_i \to bD_{i+1}\}$
For $i = n$
$P_n = \{S_n \to c_nA_n, A_n \to aA_n, A_n \to a\}$ if n is odd
$P_n = \{S_n \to c_nA_n, A_n \to bA_n, A_n \to b\}$ if n is even
$M = \{\#D_i\$c_i\# \mid 1 \le i \le n\}$
We prove the result for n even. Similar arguments hold when n is odd
For n even, the derivation is as follows

$$\begin{aligned}(S_1, S_2, \cdots, S_n) &\Rightarrow (c_1A_1, c_2A_2, \cdots, c_nA_n)\\ &\Rightarrow^+ (c_1a^{m-1}A_1, c_2b^{m-1}A_2, \cdots, c_nb^{m-1}A_n)\\ &\Rightarrow (c_1a^m|D_2, c_2|b^mD_3, \cdots, c_nb^m)\\ &\Rightarrow (c_1a^mb^m|D_2, c_2D_2, c_3|a^mD_3, \cdots, c_nb^m)\\ &\cdots\\ &\cdots\\ &\cdots\\ &\Rightarrow (c_1a^mb^m\cdots a^mb^m, c_2D_2, c_3D_3, \cdots, c_nD_n)\end{aligned}$$

Therefore we obtain for $n = 2, 4, 6, ...$
$YsgsL_n(REG) = \{c_1(a^mb^m)^{n/2}|m \ge 1\}$
and for $n = 1, 3, 5, ...$
$YsgsL_n(REG) = \{c_1a^m(b^ma^m)^{(n-1)/2}|m \ge 1\}$.

It can be seen that $L(\Gamma)$ is not in $YsgsL_{n-1}(REG)$ as it cannot be generated by

a $Ysgs$ language with $n-1$ components. In fact with only one component strings of the form $c_i a^m$ or $c_i b^m$ can only be generated. But when there are two components simple splicing of strings generated in the components gives strings of the form $c_i a^m b^m$. Extending this idea it is seen that the language specified needs n components. □

4 Conclusion

Using simple splicing rules, we have examined the power of splicing grammar systems with regular and CF component grammars. It remains to be seen how does this compare with parallel communicating grammar systems.

References

[1] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun, *Grammar systems: A grammatical approach to distribution and cooperation.* Gordon and Breach Science Publishers, 1994.

[2] J. Dassow and V. Mitrana, Splicing grammar systems, *Computers and Artificial Intelligence*, 15, 1996, 109-122.

[3] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology*, 49, 1987, 737-759.

[4] T. Head, Gh. Păun and D. Pixton, Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination. In "Handbook of Formal Languages" Vol.2, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, 1997, 295-360.

[5] A. Mateescu, Gh. Păun, G. Rozenberg and A. Salomaa, Simple splicing systems, *Discrete Applied Mathematics*, 84, 1-3, 1998, 145-163.

[6] Gh. Păun, On the power of splicing grammar systems, *Ann. Univ. Buc., Matem.-Inform. Series*, 45, 1, 1996, 93-106.

[7] A. Salomaa, *Formal languages*, Academic press, 1973.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 179 - 199, 2004.

Modelling Grammar Systems by Tissue P Systems Working in the Sequential Mode

Rudolf Freund, Marion Oswald

Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
{rudi,marion}@emcc.at

Abstract

We consider tissue P systems where rules are applied when moving through a channel from one cell to another one. In a very general manner (i.e., working on arbitrary objects as strings, arrays, graphs, etc.), these tissue P systems equipped with the sequential derivation mode allow for the representation of hybrid co-operating grammar systems using the classic basic derivation modes $*, t$ and $\leq k, = k, \geq k$, for $k \geq 1$, as well as the internally hybrid modes $(\geq k \wedge \leq \ell)$, for $k, \ell \in \mathbf{N}, k \leq \ell$, and $(t \wedge \leq k)$, $(t \wedge = k)$, $(t \wedge \geq k)$, for $k \geq 1$. Moreover, we also show how these tissue P systems working in the sequential mode allow for the simulation of random context grammars, too.

1 Introduction

For the many variants of P systems (introduced as membrane systems in [24]) investigated so far we refer the reader to [25] for a comprehensive overview as well as [28] for the actual state of research. We assume the reader to be familiar with the original definitions and explanations given for these models, as going into more details would go far beyond the scope of an introductory article to a new field of applying the ideas of membrane computing as this one is intended to be. In this paper, we consider a general model of tissue P systems that will allow us to model hybrid co-operating grammar systems when working in the sequential derivation mode, which result will be established for arbitrary object types, e.g., strings and arrays.

Co-operating distributed (CD) grammar systems first were introduced in [20] with motivations related to two-level grammars. Later, grammar systems became a vivid area of research after relating CD grammar systems with Artificial Intelligence (AI) notions [2], such as multi-agent systems or blackboard models for problem solving [22]. A first survey on grammar systems is given in [4]. The main idea of (hybrid) CD grammar systems is the co-operation of several components ("agents") on the same sentential form; non-deterministically, one component takes the sentential form, performs some derivation steps, and according to some specific stopping condition returns it back such that another component may continue the work.

There are several ways to formalize this collaboration. The following (derivation) modes have thoroughly been investigated in the literature:

- $\Longrightarrow^{\leq k}$: the component which takes the sentential form in order to work on it has to perform at most k derivation steps.
- $\Longrightarrow^{=k}$: the component ... has to perform exactly k derivation steps.
- $\Longrightarrow^{\geq k}$: the component ... has to perform at least k derivation steps.
- $\Longrightarrow^{t}$: the component ... has to perform as many derivation steps as possible.
- $\Longrightarrow^{*}$: the component ... can perform arbitrarily many derivation steps.

In CD grammar systems, all components work according to the same mode. In hybrid CD grammar systems introduced by Mitrana and Păun in [21, 23], different components may work in different modes. In a series of papers on hybrid modes in CD grammar systems, the internally hybrid modes $(\geq k \wedge \leq \ell)$, for $k, \ell \in \mathbf{N}, k \leq \ell$, and $(t \wedge \leq k)$, $(t \wedge = k)$, $(t \wedge \geq k)$, for $k \geq 1$, were considered: [7] introduced hybrid modes in CD array grammar systems as a natural specification tool for array languages, [15] investigated accepting CD grammar systems with hybrid modes, while [14] as well as [1] stressed descriptional complexity issues. In [12], results on the combination of the modes t and $\geq k$ were presented, in [13], results on the combination of the modes t with the modes $\leq k$ and $= k$ were presented. Most parts of [12, 13, 14] are contained in the report [9].

The internally hybrid modes informally can be described as follows:

- $\Longrightarrow^{(\geq k_1 \wedge \leq k_2)}$: the component which takes the sentential form in order to work on it has to perform at least k_1 and at most k_2 derivation steps.
- $\Longrightarrow^{(t \wedge \geq k)}$: the component ... has to perform as many derivation steps as possible, and at least k steps.
- $\Longrightarrow^{(t \wedge = k)}$: the component ... has to perform as many derivation steps as possible, and exactly k steps.
- $\Longrightarrow^{(t \wedge \leq k)}$: the component ... has to perform as many derivation steps as possible, and at most k steps.

Combinations $(* \wedge f)$ for $f \in \{*, t\} \cup \{ \leq k, = k, \geq k, \mid k \in \mathbf{N} \}$ are only an alternative notation of the original mode f.

The rest of the paper is organized as follows: In the next section, we start with introducing a general model for (sequential) grammars as well as for random-context grammars and ordered grammars, and then we recall some notions for string grammars and array grammars in the general setting used in this paper. In the third section, we define a general model of grammar systems covering the variants of hybrid co-operating distributed grammar systems considered above. In the fourth section, we define the general model of tissue P systems with channel rules to be used in the following section for modelling hybrid co-operating distributed grammar systems of

arbitrary types. In the sixth section, we show how random-context grammars can be simulated by tissue P systems with channel rules. Some results for the array and the string case that can be derived from the general results proved in the preceding sections are recalled in the seventh section. An outlook to future research concludes the paper.

2 Preliminary Definitions

The set of integers is denoted by $\mathbf{Z}$, the set of non-negative integers by $\mathbf{N}_0$ and the set of positive integers by $\mathbf{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^* \setminus \{\lambda\}$ is denoted by V^+. For more details on formal language theory we refer to [6] and [30].

2.1 Grammar schemes

In the following, we shall deal with various types of objects and grammars, hence, we first define a general model of a grammar scheme:

A *grammar scheme* G is a construct

$(O, O_T, P, \Longrightarrow_G)$ where

- O is the set of *objects*;
- $O_T \subseteq O$ is the set of *terminal* objects;
- P is a finite set of *productions*;
- $\Longrightarrow_G \subseteq O \times O$ is the *derivation relation* of G induced by the productions in P.

The derivation relation $\Longrightarrow_G$ is obtained as the union of all $\Longrightarrow_p \subseteq O \times O$, i.e., $\Longrightarrow_G := \cup_{p \in P} \Longrightarrow_p$, where each $\Longrightarrow_p$ is a relation which we assume at least to be recursive. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\overset{*}{\Longrightarrow}_G$.

In the following we shall consider different types of grammar schemes depending on the components of G, especially with respect to different types of productions.

Based on grammar schemes of specific types, we now define the notion of a (sequential) grammar.

Let $G = (O, O_T, P, \Longrightarrow_G)$ be a grammar scheme. Then the pair (G, w) with $w \in O$ is called a *grammar*, w is the *axiom (start object)*.

The *language generated by* (G, w) is the set of all terminal objects (we also assume $v \in O_T$ to be decidable for every $v \in O$) derivable from the axiom, i.e.,

$$L(G, w) = \left\{ v \in O_T \mid w \overset{*}{\Longrightarrow}_G v \right\}.$$

The family of languages generated by grammars of type X is denoted by $\mathcal{L}(X)$.

In many cases, the type X of the grammar scheme allows for the following feature:

A type X of a grammar scheme is called a *type with unit rules* if for every grammar scheme $G = (O, O_T, P, \Longrightarrow_G)$ of type X there exists a grammar scheme $G' = (O, O_T, P \cup P^+, \Longrightarrow_{G'})$ of type X such that

- $P^+ = \{p^+ \mid p \in P\}$,
- for all $x \in O$, p is applicable to x if and only if p^+ is applicable to x, and
- for all $x \in O$, the application of p^+ to x - in case p^+ is applicable to x - yields x back again.

2.2 Random-context grammars and ordered grammars

Let $G = (O, O_T, P, \Longrightarrow_G)$ be a grammar scheme of type X.

A *random-context grammar scheme* G_{RC} of type X is a construct

$$(G, P', p, f, \Longrightarrow_{G_{RC}})$$

where

- P' is a subset of P;
- p is a function assigning a set of *permitting productions* from P to each production in P';
- f is a function assigning a set of *forbidden productions* from P to each production in P';
- $\Longrightarrow_{G_{RC}}$ is the derivation relation assigned to G_{RC} such that for any $x, y \in O$, $x \Longrightarrow_{G_{RC}} y$ if and only if $x \Longrightarrow y$ by some q from P' and, moreover, at least one production from $p(q)$ is applicable to x as well as no production from $f(q)$ is applicable to x.

A *random-context grammar* is a pair (G_{RC}, w), where $w \in O$ is the *axiom*. A random-context grammar (scheme) is called a *grammar (scheme) with permitting context* if $f(q) = \emptyset$ for every $q \in P'$ and a *grammar (scheme) with forbidden context* if $p(q) = \emptyset$ for every $q \in P'$. The families of languages generated by random-context grammars, by grammars with permitting context, and by grammars with forbidden context of type X are denoted by $\mathcal{L}(X\text{-}RC)$, $\mathcal{L}(X\text{-}pC)$, and $\mathcal{L}(X\text{-}fC)$, respectively. Obviously, for any arbitrary type X, $\mathcal{L}(X\text{-}yC) \subseteq \mathcal{L}(X\text{-}RC)$ for $y \in \{f, p\}$.

An *ordered grammar scheme* G_O of type X is a construct

$$((O, O_T, P, \Longrightarrow_G), <, \Longrightarrow_{G_O})$$

where $<$ is a partial order relation on the productions in P and $\Longrightarrow_{G_O}$ is the derivation relation assigned to G_O such that for any $x, y \in O$, $x \Longrightarrow_{G_O} y$ if and only if $x \Longrightarrow y$ by some q from P and, moreover, no production r with $r > q$ is applicable to x.

An *ordered grammar* is a pair (G_O, w), where $w \in O$ is the *axiom*. The family of languages generated by ordered grammars is denoted by $\mathcal{L}(X\text{-}O)$.

In the general setting used in this paper, the model of ordered grammars is very much related with the model of grammars with forbidden context:

Theorem 2.1. *For every ordered grammar G_O of type X we can construct a grammar with forbidden context G_{fC} of type X such that $L(G_O) = L(G_{fC})$, i.e., $\mathcal{L}(X\text{-}O) \subseteq \mathcal{L}(X\text{-}fC)$ for any arbitrary type X.*

Proof. Let $G = (O, O_T, P, \Longrightarrow_G)$ be a grammar scheme of type X and let $(G, <, \Longrightarrow_{G_O})$ be an ordered grammar of type X. Then we construct the grammar with forbidden context $\left(G, P, f, \Longrightarrow_{G_{fC}}\right)$ of type X by defining

$$f(q) := \{p \mid p > q\}$$

for all $q \in P$. Obviously, by this construction $L(G_O) = L(G_{fC})$. □

The reverse inclusion only holds true for grammars with forbidden context that are not using additional rules in the sets of forbidden rules:

Theorem 2.2. *For every grammar with forbidden context G_{fC} of type X with*

$$\left((O, O_T, P, \Longrightarrow_G), P, f, \Longrightarrow_{G_{fC}}\right)$$

we can construct an ordered grammar G_O of type X such that $L(G_{fC}) = L(G_O)$.

Proof. We construct the ordered grammar $((O, O_T, P, \Longrightarrow_G), <, \Longrightarrow_{G_O})$ by defining, for all $q \in P$,

$$p > q \text{ if and only if } p \in f(q).$$

Obviously, by this construction $L(G_{fC}) = L(G_O)$. □

2.3 String grammars

A *string grammar scheme* usually is defined as a construct

(N, T, P) where

- N is the alphabet of *non-terminal symbols*;
- T is the set of *terminal* symbols, $N \cap T = \emptyset$;
- P is a finite set of *productions* of the form $u \to v$ with $u \in V^+$ and $v \in V^*$, where $V := N \cup T$.

In the general notion of the preceding subsection, a string grammar scheme G now is represented as

$$((V \cup T)^*, T^*, P, \Longrightarrow_G)$$

where the derivation relation for $u \to v \in P$ is defined as usual by $xuy \Longrightarrow_{u \to v} xvy$ for all $x, y \in V^*$, thus yielding the well-known derivation relation $\Longrightarrow_G$ for the string grammar scheme G. A *string grammar* then is a pair (G, S) where $S \in V - T$ is the start symbol.

As special types of string grammars we consider string grammars with arbitrary productions, context-free productions of the form $A \to v$ with $A \in N$ and $v \in V^*$, λ-free context-free productions of the form $A \to v$ with $A \in N$ and $v \in V^+$, (right-)regular productions of the form $A \to v$ with $A \in N$ and $v \in TV \cup T$, the corresponding types of grammars denoted by $ENUM$, CF, $CF_{-\lambda}$, and REG, thus yielding the families of languages $\mathcal{L}(ENUM)$, i.e., the family of recursively enumerable languages, as well as $\mathcal{L}(CF)$, $\mathcal{L}(CF_{-\lambda})$, and $\mathcal{L}(REG)$, i.e., the families of context-free, λ-free context-free, and regular languages, respectively. Observe that the types $ENUM$, CF, and $CF_{-\lambda}$ are types with unit rules (of the form $w \to w$ for $w \to v \in P$), whereas the type REG (in the definition given above) is not a type with unit rules (therefore, we often allow regular productions to be of the general form $A \to v$ with $A \in N$ and $v \in T^*V \cup T^*$).

2.4 Arrays and array grammars

In this subsection we introduce the basic notions for n-dimensional arrays and array grammar schemes and array grammars (e.g., see [16], [29], [31]).

Let $d \in \mathbf{N}$. Then a *d-dimensional array* $\mathcal{A}$ over an alphabet V is a function $\mathcal{A}: \mathbf{Z}^d \to V \cup \{\#\}$, where *shape* $(\mathcal{A}) = \{v \in W \mid \mathcal{A}(v) \neq \#\}$ is finite and $\# \notin V$ is called the *background* or *blank-symbol*. We usually write $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in \textit{shape}(\mathcal{A})\}$.

The set of all d-dimensional arrays over V is denoted by V^{*d}. The *empty array* in V^{*d} with empty shape is denoted by Λ_d. Moreover, we define $V^{+d} = V^{*n} \setminus \{\Lambda_d\}$.

Let $v \in \mathbf{Z}^d$, $v = (v_1, \ldots, v_d)$. The *translation* $\tau_v : \mathbf{Z}^d \to \mathbf{Z}^d$ is defined by $\tau_v(w) = w + v$ for all $w \in \mathbf{Z}^d$, and for any array $\mathcal{A} \in V^{*d}$ we define $\tau_v(\mathcal{A})$, the corresponding d-dimensional array translated by v, by $(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v)$ for all $w \in \mathbf{Z}^d$. The vector $(0, \ldots, 0) \in \mathbf{Z}^d$ is denoted by Ω_d.

A *d-dimensional array production* p over V is a triple $(W, \mathcal{A}_1, \mathcal{A}_2)$, where $W \subseteq \mathbf{Z}^d$ is a finite set and $\mathcal{A}_1$ and $\mathcal{A}_2$ are mappings from W to $V \cup \{\#\}$ such that *shape* $(\mathcal{A}_1) \neq \emptyset$. We say that the array $\mathcal{B}_2 \in V^{*d}$ is *directly derivable* from the array $\mathcal{B}_1 \in V^{*d}$ by the d-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$, i.e., $\mathcal{B}_1 \Longrightarrow_p \mathcal{B}_2$, if and only if there exists a vector $v \in \mathbf{Z}^d$ such that $\mathcal{B}_1(w) = \mathcal{B}_2(w)$ for all $w \in \mathbf{Z}^d \setminus \tau_v(W)$ as well as $\mathcal{B}_1(w) = \mathcal{A}_1(\tau_{-v}(w))$ and $\mathcal{B}_2(w) = \mathcal{A}_2(\tau_{-v}(w))$ for all $w \in \tau_v(W)$, i.e., the sub-array of $\mathcal{B}_1$ corresponding to $\mathcal{A}_1$ is replaced by $\mathcal{A}_2$, thus yielding $\mathcal{B}_2$.

A *d-dimensional array grammar scheme* is a grammar

$$\left((N \cup T)^{*d}, T^{*d}, P, \Longrightarrow_G\right) \text{ where}$$

- N is the alphabet of *non-terminal symbols*;
- T is the set of *terminal* symbols, $N \cap T = \emptyset$;
- P is a finite set of *array productions* over V, $V := N \cup T$;

- $\Longrightarrow_G$ is the derivation relation induced by the array productions in P according to the explanations given above, i.e., for arbitrary $\mathcal{B}_1, \mathcal{B}_2 \in V^{*d}$, $\mathcal{B}_1 \Longrightarrow_G \mathcal{B}_2$, if and only if there exists a d-dimensional array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in P such that $\mathcal{B}_1 \Longrightarrow_p \mathcal{B}_2$.

A d-dimensional array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in P is called

- *#-context-free*, if *shape* $(\mathcal{A}_1) = \{\Omega_d\}$; a d-dimensional #-context-free array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in the following will be represented in the form $\mathcal{A}_1(\Omega_d) \to \mathcal{A}_2(\Omega_d)\{(v, \mathcal{A}_2(v)) \mid v \in U\}$ with $U = W - \{\Omega_d\}$.
- *context-free*, if it is #-context-free and *shape* $(\mathcal{A}_1) \subseteq$ *shape* $(\mathcal{A}_2)$;
- *strictly context-free*, if it is context-free and *shape* $(\mathcal{A}_2) = W$;
- *regular*, if it is strictly context-free and of one of the following forms:
  1. $A \to b$, $A \in N$, $b \in T$;
  2. $A \to b\{(v, C)\}$, $A, C \in N$, $b \in T$, and v is a vector with norm 1.

A d-dimensional #-context-free array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in the following will be represented in the form $\mathcal{A}_1(\Omega_d) \to \mathcal{A}_2(\Omega_d)\{(v, \mathcal{A}_2(v)) \mid v \in U\}$ with $U = W - \{\Omega_d\}$.

A *d-dimensional array grammar* is a pair (G, w) grammar where G is a d-dimensional array grammar scheme and $\{(v_0, S)\}$ with $S \in N$ and $v_0 \in \mathbf{Z}^d$ is the *start array (axiom)*.

An array grammar (scheme) is said to be of type d-$ENUMA$, d-#-CFA, d-CFA, d-$SCFA$, d-$REGA$, respectively, if every array production in P is of the corresponding type, i.e., a d-dimensional arbitrary, #-context-free, context-free, strictly context-free or regular array production, respectively. The corresponding families of d-dimensional array languages of type X are denoted by $\mathcal{L}(X)$. $\mathcal{L}(d\text{-}ENUMA)$ is the family of recursively enumerable d-dimensional array languages. Observe that only the types d-$ENUMA$, d-#-CFA, and d-CFA are types with unit rules.

3 A General Model for Grammar Systems

In this section we define a general model of grammar systems covering the variants of hybrid co-operating distributed grammar systems considered in this paper.

Let

$$G = (O, O_T, P, \Longrightarrow_G)$$

be a grammar scheme of arbitrary type X. A *hybrid co-operating distributed grammar scheme* G_{HCD} of type X is a construct

$$\left(G, P_1, ..., P_n, \Longrightarrow_{G_{HCD}}^{(f_1,...,f_n)}\right) \text{ where}$$

predicate	definition
$Q(=k,m,i,y)$	$m=k$
$Q(\leq k,m,i,y)$	$m\leq k$
$Q(\geq k,m,i,y)$	$m\geq k$
$Q(*,m,i,y)$	$m\geq 0$
$Q(t,m,i,y)$	$\neg\exists z(y\Rightarrow_i z)$
$Q((f_1\wedge f_2),m,i,y)$	$Q(f_1,m,i,y)\wedge Q(f_2,m,i,y)$

Table 1: Definition of predicate Q for $k\in\mathbf{N}$ and $f_1,f_2\in B$

- $P_i\subseteq P$, $1\leq i\leq n$, $n\geq 1$, are *tables* of productions;
- $\Longrightarrow_{G_{HCD}}^{(f_1,\ldots,f_n)}$ is the derivation relation for G_{HCD}, which is defined as follows: For two objects $x,y\in O$ define $x\Longrightarrow_i y$ if and only if y can be derived from x by applying a production from P_i according to the derivation relation $\Longrightarrow_G$; $x\overset{m}{\Longrightarrow}_i y$ stands for a derivation in m, $m\geq 0$, such derivation steps. Now define the *classic basic (derivation) modes* $B=\{*,t\}\cup\{\leq k,=k,\geq k\mid k\in\mathbf{N}\}$ and let $D=B\cup\{(\geq k\wedge\leq\ell)\mid k,\ell\in\mathbf{N},k\leq\ell\}\cup\{(t\wedge\leq k),(t\wedge=k),(t\wedge\geq k)\mid k\in\mathbf{N}\}$. For $f\in D$, we first define a predicate Q (see Table 1) and then the relation $\Longrightarrow_i^f$ by $x\Longrightarrow_i^f y$ if and only if $\exists m\geq 0:(x\overset{m}{\Longrightarrow}_i y\wedge Q(f,m,i,y))$. For two arbitrary objects $x,y\in O$, now $x\Longrightarrow_{G_{HCD}}^{(f_1,\ldots,f_n)} y$ if and only if for some i, $1\leq i\leq n$, $x\Longrightarrow_i^{f_i} y$.

A *hybrid co-operating distributed grammar system* G_{HCD} (HCD grammar system) of type X is a pair (G_{HCD},w) where $w\in O$ is the axiom. The language *generated* by a HCD grammar system is defined as:

$$L(G_{HCD}):=\Big\{u\in O_T\mid w\Longrightarrow_{i_1}^{f_{i_1}} w_1\ldots\Longrightarrow_{i_m}^{f_{i_m}} w_m=u \quad 1\leq i_j\leq n,1\leq j\leq m,m\geq 1\}$$

If $F\subseteq D$, then the family of languages generated by HCD grammar systems of type X with degree at most n, each component working in one of the modes contained in F, is denoted by $\mathcal{L}(HCD_n,X,F)$. In a similar way, we write $\mathcal{L}(HCD_*,X,F)$ when the number of components is not restricted. If F is a singleton $\{f\}$, we simply write $\mathcal{L}(HCD_n,X,f)$, where $n\in\mathbf{N}\cup\{*\}$; in that case, the HCD grammar system is called a CD grammar system, and we also write $L_f(G)$ instead of $L(G)$ to denote the language generated by the CD grammar system G_{CD} in the mode f.

Observe that in the string case usually a hybrid co-operating distributed grammar system $G_{HCD}=\Big((V^*,T^*,P,\Longrightarrow_G),P_1,\ldots,P_n,\Longrightarrow_{G_{HCD}}^{(f_1,\ldots,f_n)},w\Big)$ is written in the form

$$(V-T,T,(P_1,f_1),\ldots,(P_n,f_n),w)$$

for some $w\in V-T$.

4 A General Model of Tissue P Systems with Channel Rules

In this section we define the general model of tissue P systems in which we are going to represent the different variants of grammar systems considered in this paper; it is based on the model of tissue-like P systems with channel states introduced in [17] and works in the sequential derivation mode (see [18] and [19]).

Let

$$G = (O, O_T, P, \Longrightarrow_G)$$

be a grammar scheme of arbitrary type X. *A tissue P system* (of degree $m \geq 1$) *with channel rules (tP system* for short) Π of type $X\,(-, 0, +)$ is a construct

$$\left(G, m, W, syn, \left(R_{(i,j)}\right)_{(i,j)\in syn}, i_o\right),$$

where

- m is the number of *cells* assumed to be labelled with $1, 2, \ldots, m$;
- $W \subseteq O^m$ are m strings over O representing the *initial finite multisets* of objects present in the m cells of the system (we here do not consider communication with the environment);
- $syn \subseteq \{(i, j) \mid i, j \in \{1, 2, \ldots, m\}\}$ is the set of *links* (also called *synapses* or *channels*, between two cells);
- $R_{(i,j)}$ is of the form R, $+R$ or $-R$, where $R \subseteq P$ is a finite set of *rules* (we say that $R_{(i,j)}$ is associated with the channel $(i, j) \in syn$); syn and $\left(R_{(i,j)}\right)_{(i,j)\in syn}$ represent the *connection graph* of Π;
- $i_o \in \{1, 2, \ldots, m\}$ is the *output* cell.

The computation starts with the configuration specified by W. In each time unit, a computation step takes place; in the *sequential derivation mode* ($\Longrightarrow_{\Pi}^{sequ}$) we first choose a synapse (i, j) such that,

R if $R_{(i,j)} = R$ and the application of a rule from R applied to some object x in cell i yields y, then x is removed from cell i and y is added in cell j, or

$+R$ if $R_{(i,j)} = +R$ and if each rule from R can be applied to some object x in cell i, then this object x can move from cell i to cell j remaining unchanged, or

$-R$ if $R_{(i,j)} = -R$ and if no rule from R can be applied to x, then this object x can move from cell i to cell j remaining unchanged.

Observe that we also allow $R = \emptyset$, i.e., a transition via $-\emptyset$ is always possible (and a transition via $+\emptyset$ is never possible and therefore makes no sense). The reflexive and transitive closure of the derivation relation $\Longrightarrow_{\Pi}^{sequ}$ is denoted by $\overset{*}{\Longrightarrow}_{\Pi}^{sequ}$. If none of the variants of a derivation step described above is possible for any of the synapses, then the derivation stops (*halts*), yet in contrast to many other variants of (tissue) P systems we do not only consider terminal objects in a halting computation as results, instead the results of a computation are described by the terminal objects from O_T present in cell i_o at any step during an arbitrary computation, i.e., the *language generated by* Π *in the sequential derivation mode* is defined as

$$L(\Pi, sequ) = \left\{ u \in O_T \mid W \overset{*}{\Longrightarrow}_{\Pi}^{f} U, u = pr_{i_0}(U) \right\};$$

the configuration of Π can be described by a vector whose components are the contents of all m cells i; pr_i is a projection yielding the contents of cell i. The family of languages generated by tP systems of type $X(-,0,+)$ is denoted by $\mathcal{L}(tP, X(-,0,+))$.

If no synapse $R_{(i,j)} \in syn$ in $\Pi = \left(G, m, W, syn, (R_{(i,j)})_{(i,j)\in syn}, i_o\right)$ is of the form $+R$, then we call Π a tP system of type $X(-,0)$, if no synapse $R_{(i,j)} \in syn$ in Π is of the form $-R$, then we call Π a tP system of type $X(0,+)$, and if all synapses are of the form R only, then we call Π a tP system of type $X(0)$. The families of languages generated by tP systems of type $X(-,0)$, $X(+,0)$, and $X(0)$ are denoted by $\mathcal{L}(tP, X(-,0))$, $\mathcal{L}(tP, X(0,+))$, and $\mathcal{L}(tP, X(0))$, respectively.

5 Tissue P Systems and Grammar Systems

In this section we give a purely structural proof for the main result saying that grammar systems of arbitrary type can be modelled by tissue P systems of the same type working in the sequential derivation mode.

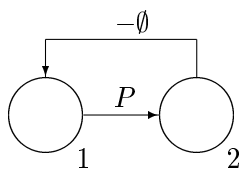

Figure 1: $*$-mode, $=$ 1-mode, $\geq$ 1-mode, $\leq k$-mode

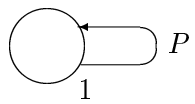

Figure 2: $*$-mode in extended variant

Theorem 5.1. *For every HCD grammar system G_{HCD} of type X we can construct a tP system of type $X(-,0)$ working in the sequential derivation mode such that*

$L(G_{HCD}) = L(\Pi, sequ)$, i.e., $\mathcal{L}(HCD_*, X, F) \subseteq \mathcal{L}(tP, X(-,0))$ *for any arbitrary non-empty set of derivation modes* F.

Proof (sketch). Let $G = (O, O_T, P, \Longrightarrow_G)$ be a grammar scheme of type X and let

$$G_{HCD} = \left(G, P_1, ..., P_n, \Longrightarrow_{G_{HCD}}^{(f_1,...,f_n)}, w\right)$$

be a HCD grammar system of type X. Then we construct the corresponding tP system

$$\Pi = (G, m, W, syn, (R_{(i,j)})_{(i,j)\in syn}, i_o)$$

as follows:

We take cell 1 as starting point of our derivations as well as the output cell i_0, too, i.e.

- $W = (w, \emptyset, ..., \emptyset)$ and
- $i_0 = 1$.

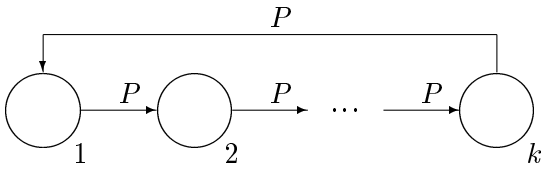

Figure 3: $= k$-mode

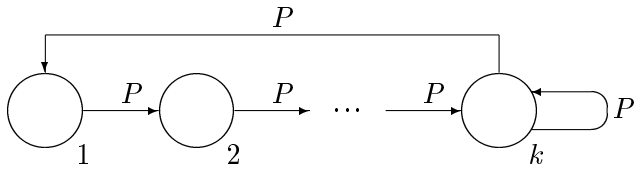

Figure 4: $\geq k$-mode

For each pair (P_i, f_i), $1 \leq i \leq n$, we now construct a number of cells together with the corresponding synapses; instead of giving formal descriptions, we describe the structure of the necessary cells and their synapses by illustrative figures; observe that the "starting point" of all simulations is cell 1.

The derivation modes $*$, $= 1$, ≥ 1, ≤ 1, and even $\leq k$ for $k \geq 2$ have the same effect and can be simulated in Π as described in Figure 1, i.e., for a pair (P, f) with $f \in \{*, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$ we only need a simple loop and only one additional cell (as a technical detail we mention that we need to go back with a synapse having assigned $-\emptyset$ because each synapse may carry only one $-R$ or R; we could avoid this by allowing finite sets of such objects assigned to a synapse, which would yield the even simpler solution depicted in Figure 2.

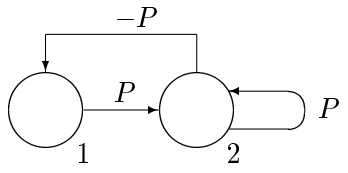

Figure 5: t-mode

For a pair (P, f) with $f \in \{= k \mid k \geq 2\}$ we need a simple loop with $k-1$ additional cells and the structure as depicted in Figure 3.

For a pair (P, f) with $f \in \{\geq k \mid k \geq 2\}$ we again need a simple loop with $k-1$ additional cells as well as an additional loop at cell k and the structure as depicted in Figure 4.

A pair (P, t) with the t-mode needs only one additional cell, but a nontrivial synapse with $-P$ as depicted in Figure 5.

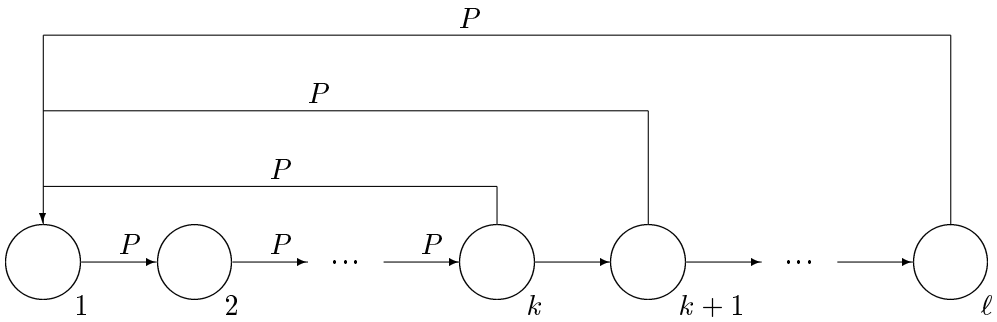

Figure 6: $(\geq k \wedge \leq \ell)$-mode, $k < l$

Suitable simulations of the internally hybrid modes $(\geq k \wedge \leq \ell)$, for $k, \ell \in \mathbf{N}, k < \ell$ (observe that for $k = l$ the derivation mode $(\geq k \wedge \leq \ell)$ coincides with the mode $= k$) and $(t \wedge \leq k)$, $(t \wedge = k)$, $(t \wedge \geq k)$, for $k \geq 1$, are illustrated in Figures 6, 7, 8, and 9.

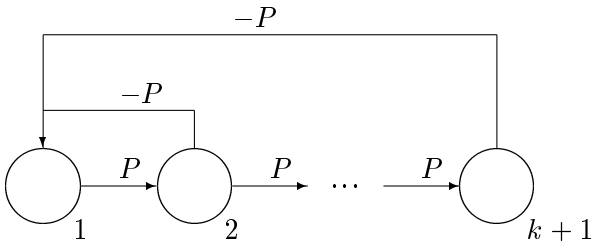

Figure 7: $(t \wedge \leq k)$-mode

In a depictive way, all the figures explained above show how the tP system Π can easily be constructed from the given HCD grammar system G_{HCD} of type X. Moreover, from the given construction we immediately infer $L(G_{HCD}) = L(\Pi, sequ)$; again we should like to stress that this result holds true for any arbitrary type X.□

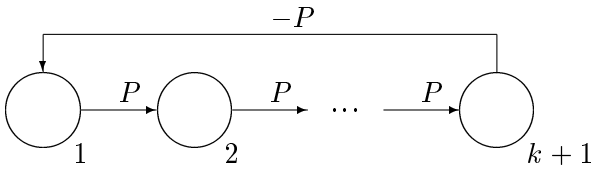

Figure 8: $(t\wedge = k)$-mode

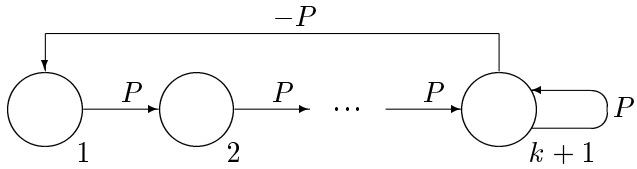

Figure 9: $(t\wedge \geq k)$-mode

Now let X be a type with unit rules. Then except for the t-mode and the hybrid modes containing the t-mode, tP systems of type $X\,(0)$ are sufficient:

Corollary 5.2. *For every HCD grammar system G_{HCD}*

$$G_{HCD} = \left((O, O_T, P, \Longrightarrow_G), P_1, ..., P_n, \Longrightarrow_{G_{HCD}}^{(f_1,...,f_n)}, w\right)$$

of a type X with unit rules such that $\{f_1, ..., f_n\}$ is a subset of

$$\{*\} \cup \{\leq k, = k, \geq k \mid k \geq 1\} \cup \{(\geq k \wedge \leq \ell) \mid k, \ell \in \mathbf{N}, k \leq \ell\}$$

we can construct a tP system of type $X\,(0)$ working in the sequential derivation mode such that $L\,(G_{HCD}) = L\,(\Pi, sequ)$, i.e.,

$$\mathcal{L}\,(HCD_*, X, F) \subseteq \mathcal{L}\,(tP, X\,(0))$$

for any arbitrary non-empty set of derivation modes

$$F \subseteq \{*\} \cup \{\leq k, = k, \geq k \mid k \geq 1\} \cup \{(\geq k \wedge \leq \ell) \mid k, \ell \in \mathbf{N}, k \leq \ell\}.$$

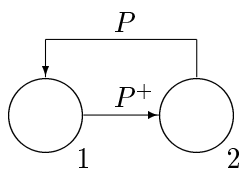

Figure 10: non-t-modes with unit rules

Proof (sketch). In the construction of the tP system in the proof of Theorem 5.1, the $*$-mode now can be simulated as shown in Figure 10, where P^+ is the set of unit rules corresponding to the set of rules P. □

For more restricted sets of derivation modes, the result of the preceding corollary even holds for arbitrary types X; moreover, we only need one synapse as shown in Figure 11:

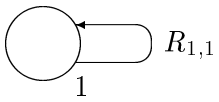

Figure 11: $*$-mode with unit rules

Corollary 5.3. *For every HCD grammar system* G_{HCD}

$$G_{HCD} = \left((O, O_T, P, \Longrightarrow_G), P_1, ..., P_n, \Longrightarrow_{G_{HCD}}^{(f_1,...,f_n)}, w\right)$$

of an arbitrary type X *such that*

$$\{f_1, ..., f_n\} \subseteq \{*, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$$

we can construct the tP system

$$\Pi = \left((O, O_T, P, \Longrightarrow_G), 1, W, \{1\}, R_{(1,1)}, 1\right)$$

of type $X\,(0)$ *with* $R_{(1,1)} = \cup_{i=1}^{n} P_i$ *working in the sequential derivation mode such that* $L\,(G_{HCD}) = L\,(\Pi, sequ)$*; hence,*

$$\mathcal{L}\,(HCD_*, X, F) = \mathcal{L}\,(X) \subseteq \mathcal{L}\,(tP, X\,(0))$$

for any arbitrary non-empty set of derivation modes

$$F \subseteq \{*, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}\,.$$

For the inverse inclusion relations we just mention that the tP systems investigated in this section exactly characterize the corresponding variants of HCD grammar systems when having the specific connection graphs, i.e., the structures of synapses and their connections, as described above, the other cells always being arranged around the output cell, which also carries the only input object.

6 Tissue P Systems and Random-Context Grammars

To prove our next result in the general case, we need channel rules $R_{(i,j)}$ of the form $+R$ (which allows us to check for permitting context or to check for competence as this is called by Jürgen Dassow in [5]).

Theorem 6.1. *For every random-context grammar* G_{RC} *of type* X *we can construct a tP system* Π *of type* $X\,(-, 0, +)$ *working in the sequential derivation mode such that* $L(G_{RC}) = L(\Pi, sequ)$*, i.e.,* $\mathcal{L}\,(X\text{-}RC) \subseteq \mathcal{L}\,(tP, X\,(-, 0, +))\,.$

Proof. Let $G = (O, O_T, P, \Longrightarrow_G)$ be a grammar scheme of type X and let

$$G_{RC} = (G, P', p, f, \Longrightarrow_{G_{RC}})$$

be a random-context grammar of type X with $P' = \{q_1, ..., q_n\}$. Then we construct the corresponding tP system

$$\Pi = (G, 2n+1, W, syn, (R_{(i,j)})_{(i,j)\in syn}, 1)$$

by using the simulation described in Figure 12 for every production q_i in P', i.e.:

- $syn = \{(1, i+1), (i+1, 2i+1), (2i+1, 1) \mid 1 \leq i \leq n\}$;
- $R_{(1,i+1)} = +p(q_i)$, $1 \leq i \leq n$;
- $R_{(i+1,2i+1)} = -f(q_i)$, $1 \leq i \leq n$;
- $R_{(2i+1,1)} = \{q_i\}$, $1 \leq i \leq n$.

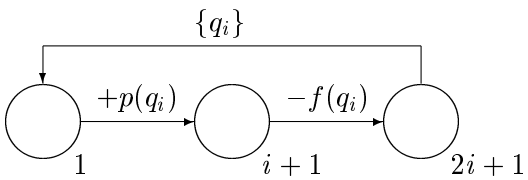

Figure 12: Π simulating G_{RC}

Observe that even in case $f(q_i) = \emptyset$ the simulation still works correctly. In the case $p(q_i) = \emptyset$ we have to omit one cell and the channel rule $+p(q_i)$ (see Figur 13); another possibility is to set $p(q_i) := \{q_i\}$ instead, which is always possible, too. Obviously, by this construction $L(G_{RC}) = L(\Pi, sequ)$; again we should like to stress that this result holds true for any arbitrary type X. □

The following corollaries for grammars with forbidden context G_{fC} and grammars with permitting context G_{pC} are immediate consequences of the construction given in the proof of the preceding theorem:

Corollary 6.2. *For any arbitrary type X, $\mathcal{L}(X\text{-}fC) \subseteq \mathcal{L}(tP, X(-, 0))$.*

Proof. The simulation now works as depicted in Figure 13, i.e., we construct the corresponding tP system

$$\Pi = (G, n+1, W, syn, (R_{(i,j)})_{(i,j)\in syn}, 1)$$

with

- $syn = \{(1, i+1), (i+1, 1) \mid 1 \leq i \leq n\}$;
- $R_{(1,i+1)} = -f(q_i)$, $1 \leq i \leq n$;
- $R_{(i+1,1)} = \{q_i\}$, $1 \leq i \leq n$. □

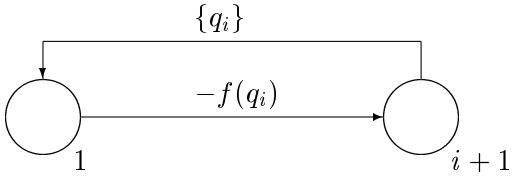

Figure 13: Π simulating G_{fC}

Corollary 6.3. *For any arbitrary type X, $\mathcal{L}(X\text{-}pC) \subseteq \mathcal{L}(tP, X(0,+))$.*

Proof. The simulation now works as depicted in Figure 14, i.e., we construct the corresponding tP system

$$\Pi = (G, n+1, W, syn, (R_{(i,j)})_{(i,j)\in syn}, 1)$$

with

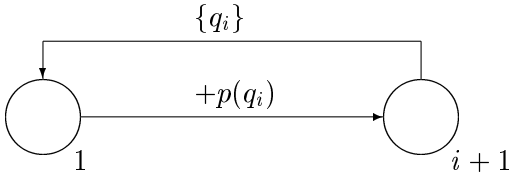

Figure 14: Π simulating G_{pC}

- $syn = \{(1, i+1), (i+1, 1) \mid 1 \leq i \leq n\}$;
- $R_{(1,i+1)} = +p(q_i)$, $1 \leq i \leq n$;
- $R_{(i+1,1)} = \{q_i\}$, $1 \leq i \leq n$.

As a special technical proof detail we mention that without loss of generality we may assume $q_i \in p(q_i)$. □

If the underlying type X is a type with unit rules, we do not need channel rules $R_{(i,j)}$ of the form $+R$ as is shown in Figure 15 and Figure 16 - observe that in case of $p(q_i) = \emptyset$ we have to take $p(q_i)^+ := \{q_i\}^+$ ($= \{q_i{}^+\}$); hence, we obtain the following results:

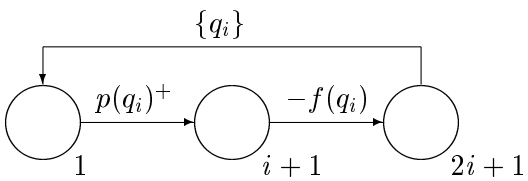

Figure 15: Π simulating G_{RC} with unit rules

Corollary 6.4. *For any type X with unit rules, $\mathcal{L}(X\text{-}RC) \subseteq \mathcal{L}(tP, X(-,0))$.*

Corollary 6.5. *For any type X with unit rules, $\mathcal{L}(X\text{-}pC) \subseteq \mathcal{L}(tP, X(0))$.*

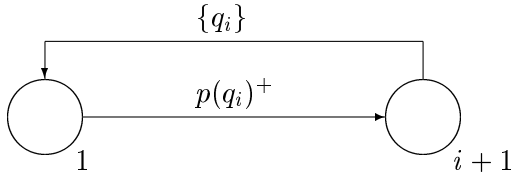

Figure 16: Π simulating G_{pC} with unit rules

7 Tissue P Systems Working on Strings and Arrays

Based on the general result presented in the preceding section, we can immediately infer the corresponding results for the types of string and array grammars defined in the second section.

For example, in the string case we cite the following two results from Theorem 19 and Corollary 21 in [13]:

Corollary 7.1. *If* $\emptyset \neq F \subseteq \{ = k, \geq k, (\geq k \wedge \leq \ell), (t \wedge \geq k) \mid 2 \leq k \leq \ell \}$, *then*

$$\mathcal{L}(HCD_*, CF, F \cup \{(t \wedge = 1)\}) = \mathcal{L}(ENUM).$$

Corollary 7.2. *Let* $\emptyset \neq F \subseteq \{*, t\} \cup \{ \leq k, = k, \geq k, (t \wedge \geq k) \mid k \geq 1 \} \cup \{ (\geq k \wedge \leq \ell) \mid k, \ell \geq 1, k \leq \ell \}$. *Then, for every* $m \geq 2$,

$$\mathcal{L}(HCD_*, CF, F \cup \{(t \wedge = m)\}) = \mathcal{L}(ENUM).$$

Obviously, from these results we immediately obtain the following one due to Theorem 5.1:

Corollary 7.3. *For every recursively enumerable string language* L *we can construct a tP system* Π *of type* CF *such that* $L(\Pi, sequ) = L$, *i.e.,*

$$\mathcal{L}(ENUM) = \mathcal{L}(tP(-, 0), CF).$$

The structure of the connection graph of the tP system Π in Corollary 7.3 depends on the set of derivation modes F we take according to Corollary 7.1 or Corollary 7.2.

Moreover, from Corollary 5.3 we immediately obtain the following result:

Corollary 7.4. *Let* $\emptyset \neq F \subseteq \{*, = 1, \geq 1\} \cup \{ \leq k \mid k \geq 1 \}$. *Then*

$$\mathcal{L}(HCD_*, X, F) = \mathcal{L}(X)$$

for $X \in \{ENUM, CF, CF_{-\lambda}\} \cup \{d\text{-}ENUMA, d\text{-}\#\text{-}CFA, d\text{-}CFA \mid d \geq 1\}$.

Several other results proved in the preceding sections carry over to the case of array grammar systems of various types (e.g., see [3]), too. Following the results elaborated in [8] and [11], we even get applications of tP systems of types 2-(#-)CFA in the area of character recognition, e.g., see [10].

A generalization of the two-dimensional case to arbitrary dimensions of a result proved in [16] shows that, for any arbitrary dimension $d \geq 1$,

$$\mathcal{L}\,(d\text{-}ENUMA) = \mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}O);$$

according to Theorem 2.1,

$$\mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}O) \subseteq \mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}fC)\,;$$

due to Corollary 6.2,

$$\mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}fC) \subseteq \mathcal{L}\,(tP, d\text{-}\#\text{-}CFA\,(-,0))\,.$$

In sum, we get the following result:

Corollary 7.5. *For any arbitrary dimension* $d \geq 1$,

$$\mathcal{L}\,(d\text{-}ENUMA) = \mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}O) =$$
$$\mathcal{L}\,(d\text{-}\#\text{-}CFA\text{-}fC) = \mathcal{L}\,(tP, d\text{-}\#\text{-}CFA\,(-,0))\,.$$

Without proof we should like to mention that for $d \in \{1,2,3\}$ we could even show that

$$\mathcal{L}\,(d\text{-}ENUMA) = \mathcal{L}\,(tP, d\text{-}\#\text{-}CFA\,(0))\,.$$

8 Summary and Future Research

We have shown that hybrid co-operating distributed grammar systems of arbitrary types equipped with arbitrary classic basic derivation modes as well as internally hybrid derivation modes can be simulated by tissue P systems with channel rules of the corresponding types when working in the sequential derivation mode and with only specific structures of the underlying connection graph. Moreover, we have shown how random-context grammars can be simulated by tissue P systems, too.

This paper has to be seen as a starting point for future research only. Many technical details remain for being investigated in a more precise way. Moreover, we have not considered other models of grammar systems, for example, parallel communicating grammar systems, as has been done by Gheorghe Păun, see [26].

Acknowledgements

The authors acknowledge many interesting and inspiring discussions with Francesco Bernardini, Erzsébet Csuhaj-Varjú, Jürgen Dassow, and Gheorghe Păun in the areas of grammar systems and membrane computing, especially during the Grammar Systems Week having taken place in Budapest at the beginning of July 2004 as well as IST-2001-32008 project "MolCoNet".

References

[1] H. Bordihn, M. Holzer: On a hierarchy of languages generated by cooperating distributed grammar systems. *Inform. Process. Lett.* **69**, 2 (1999) 59–62

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

[3] J. Dassow, R. Freund, Gh. Păun: Cooperating array grammar systems. *Int. Journ. of Pattern Recognition and Artificial Intelligence* **9**, 6 (1995) 1–25

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

[5] J. Dassow: On cooperating distributed grammar systems with competence based start and stop conditions. In: E. Csuhaj-Varjú, Gy. Vaszil (eds.): Prel. Proceedings of Grammar Systems Week 2004, MTA SZTAKI, Budapest (2004) 146–151

[6] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin (1989)

[7] H. Fernau, R. Freund: Bounded parallelism in array grammars used for character recognition. In: P. Perner, P. Wang, A. Rosenfeld (eds.): *Proceedings of Structural and Syntactical Pattern Recognition SSPR'96*, Lecture Notes in Computer Science **1121**. Springer, Berlin (1996) 40–49

[8] H. Fernau, R. Freund: Accepting array grammars with control mechanisms. In: Gh. Păun, A. Salomaa (eds.): *New Trends in Formal Languages*, Lecture Notes in Computer Science **1218**. Springer, Berlin (1997) 95–118

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

[10] H. Fernau, R. Freund, M. Holzer: Character recognition with k-head finite array automata. In: A. Amin et al. (eds.): *Proceedings of Structural and Syntactical Pattern Recognition SSPR'98*, Lecture Notes in Computer Science **1451**. Springer, Berlin (1998) 282–291

[11] H. Fernau, R. Freund, M. Holzer: Regulated array grammars of finite index. In: Gh. Păun, A. Salomaa (eds.): *Grammatical Models of Multi-Agent Systems.* Gordon and Breach, London (1999) 157–181 (Part I) and 284–296 (Part II)

[12] H. Fernau, R. Freund, M. Holzer: Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theoretical Computer Science* **259** (2001) 405–426

[13] H. Fernau, R. Freund, M. Holzer: Hybrid modes in cooperating distributed grammar systems: combining the t-mode with the modes $\leq k$ and $= k$. *Theoretical Computer Science* **299** (2003) 633–662

[14] H. Fernau, R. Freund, M. Holzer: Bounding resources in cooperating distributed grammar systems. In: S. Bozapalidis (ed.): *Proc. of the 3rd Int. Conf. Developments in Language Theory.* Aristotle University of Thessaloniki (1997) 261–272

[15] H. Fernau, M. Holzer: Accepting multi-agent systems II. *Acta Cybernetica* 12 (1996) 361–379

[16] R. Freund: Control mechanisms on #-context-free array grammars. In: Gh. Păun (ed.): *Mathematical Aspects of Natural and Formal Languages.* World Scientific Publ., Singapore (1994) 97–137

[17] R. Freund, Gh. Păun, M.J. Pérez Jiménez: Tissue-like P systems with channel states. In: [27] (2004) 206–223

[18] R.Freund: Asynchronous P systems. In: *Proceedings of WMC 5,* Milano, Italy, June 2004 (2004)

[19] R.Freund: Asynchronous P systems on arrays and strings. Accepted for presentation at *DLT 2004* (2004)

[20] R. Meersman, G. Rozenberg: Cooperating grammar systems. In: *Proceedings of Mathematical Foundations of Computer Science MFCS'78*, Lecture Notes in Computer Science **64**. Springer, Berlin (1978) 364–374

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

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

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

[24] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61,** 1 (2000) 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)

[25] Gh. Păun: *Membrane Computing: an Introduction.* Springer, Berlin (2002)

[26] Gh. Păun: Grammar systems vs. membrane computing: a preliminary approach. In: E. Csuhaj-Varjú, Gy. Vaszil (eds.): *Prel. Proc. of Grammar Systems Week 2004*, MTA Sztaki, Budapest (2004) 225–245

[27] Gh. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini (eds.): Dept. of Computer Sciences and Artificial Intelligence, *Univ. of Sevilla Tech. Report 01/2004*, Second Week on Membrane Computing, Sevilla, Spain, Feb. 2-7 (2004)

[28] The P Systems Web Page: `http://psystems.disco.unimib.it/`

[29] A. Rosenfeld: *Picture Languages.* Academic Press, Reading, MA (1979)

[30] A. Salomaa, G. Rozenberg (eds.): *Handbook of Formal Languages.* Springer, Berlin (1997)

[31] P. S.-P. Wang (ed.): *Array Grammars, Patterns and Recognizers.* World Scientific Series in Computer Science **18.** World Scientific Publ., Singapore (1989)

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 200 - 211, 2004.

A Possible Connection Between Two Theories: Grammar Systems and Concurrent Programming

María Adela Grando*

Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
mariaadela.grando@estudiants.urv.es

Victor Mitrana

Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain,
and
Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 70109 Bucharest, Romania
vmi@fll.urv.es

Abstract

The aim of this paper is to show how PC grammar systems and concurrent programs might be viewed as related models for distributed and cooperating computation. We argue that it is possible to translate a grammar system into a concurrent program, where one can make use of the Owicki-Gries theory and other tools available in the programming framework. The converse translation is also possible and this turns out to be useful when we are looking for a grammar system able to generate a given language.

In order to show this we use the language: $L_{cd} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$, called *crossed agreement language,* one of the basic non-context free constructions in natural and artificial languages. We prove, using tools from concurrent programming theory, that $L_{cd} \in NPC_3(REG)$ (non-returning PC grammar systems with regular components) solving an open problem introduced in [2].

We also discuss the absence of strategies in the concurrent programming theory to prove that $L_{cd} \notin X_2(REG)$, for $X \in \{PC, CPC, NPC, NCPC\}$, but we prove this in the grammar system framework.

Keywords: PC grammar systems, multiprogramming, Owicki-Gries theory.

1 Introduction

In the beginning of computation theory, classic computing devices were centralized, that is the computation was accomplished by one central processor. But in modern

*This work was possible thanks to the research grant "Programa Nacional para la formación del profesorado universitario", from the Ministry of Education, Culture and Sports of Spain.

computer science distributed computing systems that consist of multiple communicating processors play a major role. The reason is illustrated by the advantages of this kind of system: efficiency, fault tolerance, scalability in the relation between price and performance, etc.

Since 1960, when the concept of *concurrent programming* [1] was introduced, a huge variety of topics related to parallelism and concurrency have been defined and investigated such as operating systems, machine architectures, communication networks, circuit design, protocols for communication and synchronization, distributed algorithms, logics for concurrency, automatic verification and model checking.

The same trend was observed in classic formal language and automata theory as well. In the beginning, grammars and automata were modeling classic computing devices of one agent or processor, hence a language was generated by one grammar or recognized by one automaton. Inspired by different models of distributed systems in Artificial Intelligence, *grammar systems theory* [3] has been developed as a grammatical theory for distributed and parallel computation. More recently, similar approaches have been reported for systems of automata [10].

A grammar system is a set of grammars, working together, according to a specified protocol, in order to generate one language. There are many reasons to consider such a generative mechanism: to model distribution and parallelism, to increase the generative power, to decrease the (descriptional) complexity, etc. The crucial element here is the protocol of cooperation. The theory of grammar systems may be seen as the grammatical theory of cooperation protocols. The central problems are the functioning of systems under specific protocols and the influence of various protocols on various properties of considered systems.

One can distinguish two basic classes of grammar systems: sequential and parallel. In this paper we consider the second class, called *parallel communicating (PC) grammar system* [14]. In the next section, we recall the basic definitions related to this model as well as the basic concepts of concurrent programming and Owicki-Gries theory which is known as the first complete programming logic used for formal development of concurrent programs.

Owicki-Gries theory [11] and other strategies of programming were developed to help programmers in the analysis and design of multiprograms. In this paper we argue that grammar system theory can benefit from the tools already developed in the programming framework. For example, given a grammar system one can prove that it generates a specific language by a direct reasoning or one can translate the grammar system into a multiprogram and prove the same statement by some strategies of programming developed in the well known Owicki-Gries theory. Furthermore, we propose another approach to solve problems of the following type: *Given a language find a grammar system that generates the given language.* The strategy widely used so far is as follows: first one proposes a grammar system and then proves by means of language theory that the proposed grammar system generates indeed the given language. We give an example of how Owicki-Gries logic of programming could guide us in obtaining, *simultaneously*, the grammar systems that generates the given language and a proof that it really generates it. This new approach might be of a great benefit for the grammar systems theory. We apply this strategy for a well-known non-context-free language, namely $L_{cd} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$.

In [2] it was proved that $L_{cd} \in CPC_4(CF)$ (see the next section for notations), we improved this result showing that $L_{cd} \in NPC_5(REG)$, providing thus a solution to an open problem mentioned in that work. During the workshop, G. Păun proposed us a more economical solution with respect to the number of components of the grammar system, namely $L_{cd} \in NPC_3(REG)$, based on a similar strategy. We give here, in Section 3, the solution proposed by Păun. As we shall see in the same section, this is actually the most economical solution. The strategy consists in translating the problem of finding a non-returning, non-centralized PC grammar system Γ with regular components that generates L_{cd}, into the problem of finding a multiprogram $\mathcal{P}$, with three programs $Prog_i$, $i = 1, 2, 3$, running concurrently, which is correct with respect to the specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}\mathcal{P}\{w_1 \in L_{cd}\}.$$

Then this multiprogram is translated back into a PC grammar system Γ with three regular components, the whole behavior of Γ being similar to that of $\mathcal{P}$. Actually, this means that the language generated by Γ is included in L_{cd} but the detailed reasoning presented in the third section allows us to conclude the equality.

In this paper we also show that though PC grammar systems theory can benefit from concurrent programming theory to get positive results, the later theory cannot provide any strategy to deal with negative results of the type: *A given language cannot be generated by any grammar system of a specified type.* This kind of problems has to be attacked in the grammar system framework, with the tools available there. We exemplify this with a proof that $L_{cd} \notin NPC_2(REG)$.

2 Preliminaries

An *alphabet* is a finite and nonempty set of symbols. Any sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by λ. Further, $V^+ = V \setminus \{\lambda\}$.

For all unexplained notions the reader is referred to [15].

2.1 PC Grammar Systems

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

$$\Gamma = (N, K, T, (P_1, S_1), (P_2, S_2), (P_3, S_3), ..., (P_n, S_n)),$$

where:

- N is a nonterminal alphabet,
- T is a terminal alphabet,
- $K = \{Q_1, Q_2, ..., Q_n\}$ (the sets N, T, K are mutually disjoint),
- P_i is a finite set of rewriting rules over $N \cup K \cup T$, and $S_i \in N$, for all $1 \leq i \leq n$.

Let $V_\Gamma = N \cup K \cup T$. The sets P_i, $1 \leq i \leq n$, are called the components of the system, and the elements $Q_1, Q_2, ..., Q_n$ of K are called query symbols, the index i of Q_i points to the component P_i of Γ. A component is said to be regular if all its rules

are right-linear ones. An n-tuple $(x_1, x_2, ..., x_n)$ with $x_i \in V_\Gamma^*$ for all i, $1 \leq i \leq n$, is called a *configuration* of Γ. A configuration $(x_1, x_2, ..., x_n)$ directly yields another configuration $(y_1, y_2, ..., y_n)$ if either:

- No query symbols appears in $x_1, x_2, ..., x_n$ and then we have a componentwise derivation, $x_i \Longrightarrow y_i$ in each component P_i, $1 \leq i \leq n$ (one rule is used in each component P_i), except for the case when x_i is terminal, $x_i \in T^*$; then $x_i = y_i$, or

- Query symbols occur in some x_i. Then a *communication step* is performed: every x_i (containing query symbols) is modified by substituting x_j for each occurrence of a query symbol Q_j, providing x_j does not contain query symbols. After all words x_i have been modified, the component P_j continues its work on the current string (in the *non-returning* case) or resumes working from its axiom (in the *returning* case. The communication has priority over the effective rewriting: no rewriting is possible as long as at least one query symbol is present. If some query symbols are not satisfied at a given moment, then they have to be satisfied as soon as other query symbols have been satisfied.

If only the first component is entitled to introduce query symbols, then the system is called *centralized*.

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

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

Hence, one starts from the n-tuple of axioms, $(S_1, S_2, ...S_n)$, and proceeds by repeated rewriting and communication steps, until the component P_1 produces a terminal string. The component P_1 is called the *master* of the system.

The class of languages generated by non-centralized, centralized, non-returning non-centralized, non-returning centralized PC grammar systems with k regular components is denoted by $PC_k(REG)$, $CPC_k(REG)$, $NPC_k(REG)$, and $NCPC_k(REG)$, respectively.

2.2 Programming

2.2.1 Sequential Programming

A sequential program consists in:

– A number of declarations,
– A sequence of instructions or actions.

The actions take place one after another. That is, an action does not begin until the preceding one has ended. Because a sequential program has a sequence of actions we consider a program as a transformer of states or predicates (see, e.g., [4] and [7]), where a state $\{P\}$ describes the relationships between the variables of the systems and their values by the predicate P . Each action $\mathcal{S}$ transforms the current state of the system, called precondition of $\mathcal{S}$, to the state $\{Q\}$ which is called postcondition.

A *Hoare triple* is a sequence $\{P\}\,\mathcal{S}\,\{Q\}$, where:

– $\mathcal{S}$ is an action or instruction,
– $\{P\}$ is a state representing the precondition of $\mathcal{S}$,
– $\{Q\}$ is a state representing the postcondition of $\mathcal{S}$.

Its operational interpretation is as follows: $\{P\}\,\mathcal{S}\,\{Q\}$ is a correct Hoare triple if and only if it is true that each terminating execution of $\mathcal{S}$ that starts from a state satisfying P is guaranteed to end up in a state satisfying Q. More precisely, if $\{P\}\,\mathcal{S}\,\{Q\}$ holds and $\mathcal{S}$ starts in a state satisfying P, we can be sure that $\mathcal{S}$ either terminates in a state satisfying Q or does not terminate at all. Consequently, a program ought to be annotated in such a way that each action carries a precondition. In other words, from a logical perspective a sequential program may be viewed as a sequence of Hoare triples.

We can now formulate the concept of *local correctness* of a predicate Q in a program. We distinguish two cases:

– If Q is the initial predicate of the program, it is locally correct whenever it is implied by the precondition of the program as a whole. Also we may say that Q satisfies the hypothesis of the problem which is to be solved.

– If Q is preceded by $\{P\}\,\mathcal{S}$, i.e. by atomic action $\mathcal{S}$ with precondition P, it is locally correct whenever $\{P\}\,\mathcal{S}\,\{Q\}$ is a correct Hoare-triple.

A sequential program is *partially correct* if all its predicates are locally correct and the last predicate satisfies the requirements of the problem solved, provided that it halts. A sequential program is *totally correct* if it is partially correct and always halts.

2.2.2 Concurrent Programming

Concurrent execution or multiprogramming means that various sequential programs run simultaneously. Actions change the state of the multiprogram, so the critical question now is what happens if two overlapping actions change the same state of the multiprogram in a conflicting manner.

Now we are ready to formulate what we call the Core of the **Owicki-Gries theory** (see [10]). We consider a multiprogram annotated in such a way that the annotation provides a precondition for the multiprogram as a whole and a precondition for each action in each individual program. Then, by Owicki and Gries, this annotation is correct whenever each individual predicate is correct, i.e.:

– locally correct as described above and

– globally correct. Predicate Q in a multiprogram $\mathcal{M}$ is *globally correct* whenever for each $\{P\}\,\mathcal{S}$, i.e. for each action $\mathcal{S}$ with precondition P, taken from a program of $\mathcal{M}$, $\{P \wedge Q\}\,\mathcal{S}\,\{Q\}$ is a correct Hoare-triple.

To understand how powerful is the concurrent programming, and also how hard is to prove global correctness we give this simple example:

Example 1 *Consider this program:*

$$\begin{array}{llll} P_1: & x := & y+1; & a \\ & x := & y^2; & b \\ & x := & x-y & c \end{array}$$

If we start in an initial state $\{x = 7 \wedge y = 3\}$, it will deliver $\{x = 6 \wedge y = 3\}$ as a final state.

Example 2 *Also consider this simple program:*

$$\begin{array}{llll} P_2: & y := & x+1; & u \\ & y := & x^2; & v \\ & y := & y-x & w \end{array}$$

When started in the same initial state $\{x = 7 \wedge y = 3\}$, it yields $\{x = 7 \wedge y = 42\}$.

Now if we run these programs concurrently we will get 20 possible values for x and y. For instance, one possibility is to run the two programs as follows: a, u, b, v, w, c (the letters represent the program lines) starting from the same state and get the output $\{x = -224, y = 240\}$. While each of the individual programs is of an extreme simplicity, their composition leads to a rather complicated output. For more examples we refer to [5].

Here we can see the analogy: the components of a grammar systems are similar to the simple programs in a multiprogram. We introduce a proof that shows how to take advantage of this analogy.

3 Main Result

Theorem 1 $L_{cd} \in NPC_3(REG)$

Proof. We want to find a non-returning, non-centralized grammar system Γ with regular components that generates L_{cd}. This problem is transformed into the equivalent problem of finding a multiprogram $\mathcal{P}$ that behaves like Γ and is correct with respect to the specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge ..... \wedge (w_n = S_n) \mid n \geq 1\}\mathcal{P}\{w_1 \in L_{cd}\}.$$

The problem remained the same, but we changed the tools to solve it: instead of *induction* and *analysis by cases* available in the framework of grammar systems we used *Logic, Owicki-Gries theory* and *programming strategies* from the programming framework.

The strategy used for this proof is one frequently used for the development of programs, called *refinement of the problem* that consists in:

1. First, start with an outline of the solution, which identifies the basic principle by which the input can be transformed into the output. Define pre and post conditions for each of the subproblems that are identified as part of the solution for the hole problem.

For our problem we proposed this idea:

$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge ..... \wedge (w_n = S_n)\}$

Subproblem 1: *(Rewrite)^p, with $p \geq 1$*

$\{(w_1 = S_1) \wedge ..... \wedge (w_i = a^p S_i) \wedge ..... \wedge (w_j = c^p S_j) \wedge ..... \wedge (w_n = S_n) \wedge (p \geq 1)\}$

Subproblem 2: *(Rewrite; Communication)$^+$*

Find a way to stop the productions of *a's* and *c's*, through synchronization by communication.

$\{(w_1 = a^r N_1) \wedge ..... \wedge (w_k = c^r N_2) \wedge ..... \wedge (w_n = S_n) \wedge (r \geq 1) \wedge (N_1, N_2 \in N)\}$

Subproblem 3: *(Rewrite)*^{m}, *with* $m \geq 1$

$$\left\{ \begin{array}{c} (w_1 = a^r b^m Q_k) \wedge ..... \wedge (w_k = c^r d^{m-1} N_3) \wedge ..... \wedge (w_n = S_n) \wedge \\ (r, m \geq 1) \wedge (Q_k \in K) \wedge (N_3 \in N) \end{array} \right\}$$

Subproblem 4: *Communication*

$\{(w_1 = a^r b^m c^r d^{m-1} N_3) \wedge (r, m \geq 1) \wedge (N_3 \in N)\}$

Subproblem 5: *Rewrite*

$\{(w_1 = a^r b^m c^r d^m) \wedge (r, m \geq 1)\}$

or equivalently

$\{w_1 \in \{a^r b^m c^r d^m \wedge r \geq 1 \mid m \geq 1\}\}$

2. Now we make precise the outline indicated, refine the subproblems trying to find simultaneously the instructions that solve the subproblems and the proof of its local correctness. We also discuss the difficulties we can have when proving global correctness.

In our refinement of subproblems 1, 2, 3, 4 and 5 we proposed three programs $Prog_1$, $Prog_2$ and $Prog_3$. These programs forming the multiprogram P, run simultaneously behaving like a non-returning, non-centralized grammar system with regular productions and behave locally correctly with respect to the subproblems that we have identified in the previous step.

In the case of Subproblem 1 we propose this refinement:

$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}$

Subproblem 1: *Rewrite*^{n}, *with* $n \geq 1$

$Prog_1$ rewrites $n-1$ times S_1 to aS_1 and then rewrites S_1 to aA, $Prog_2$ rewrites $n-1$ times S_2 to cS_2 and then rewrites S_2 to cB and $Prog_3$ rewrites $n-1$ times S_3 to S_3, until it decides to finish the production of *a's* and *c's*, rewriting S_3 to Q_2.

To be sure that $w_2 = c^n B$ when $Prog_3$ introduces Q_2, $Prog_3$ should not be able to rewrite S_2, and after $Prog_2$ introduces B it should rewrite it for another nonterminal and not introduce B anymore.

The reason why $w_1 = a^n A$ and $w_1 \neq a^n S_1$ is that this is the only possibility that do not lead to deadlock, as the states of the next subproblem show.

$\{(w_1 = a^n A) \wedge (w_2 = c^n B) \wedge (w_3 = Q_2) \wedge (n \geq 1)\}$

For Subproblem 2 we propose this sequence of rewritings and communications as a refinement:

$\{(w_1 = a^n A) \wedge (w_2 = c^n B) \wedge (w_3 = Q_2) \wedge (n \geq 1)\}$

Subproblem 2 $\begin{cases} \textit{Communication} \\ \{w_1 = a^n A \wedge w_2 = c^n B \wedge w_3 = c^n B \wedge n \geq 1\} \\ \textit{Rewrite} \\ Prog1 \text{ rewrites } A \text{ to } A', \ Prog2 \text{ rewrites } B \text{ to } Q_1 \text{ and} \\ Prog3 \text{ rewrites } B \text{ to } D \\ \text{We do not allow other possibility than} \\ w_1 = a^n A' \wedge w_2 = c^n Q_1 \wedge w_3 = c^n D. \\ \text{To be sure that } w_1 = a^n A' \text{ after the rewriting step,} \\ \text{we need } Prog_2 \text{ to be only defined for } A', \text{ and after } Prog_1 \\ \text{introduces } A' \text{ it should rewrite it to another} \\ \text{nonterminal and not introduce } A' \text{ anymore.} \\ \{w_1 = a^n A' \wedge w_2 = c^n Q_1 \wedge w_3 = c^n D \wedge n \geq 1\} \\ \textit{Communication} \end{cases}$

$\{(w_1 = a^n A') \wedge (w_2 = c^n a^n A') \wedge (w_3 = c^n D) \wedge (n \geq 1)\}$

In the case of Subproblem 3 this is a possible refinement:

$\{(w_1 = a^n A') \wedge (w_2 = c^n a^n A') \wedge (w_3 = c^n D) \wedge (n \geq 1)\}$

Subproblem 3: *Rewrite*$^{m+1}$, *with* $m \geq 1$

$Prog_1$ rewrites A' to A'' and rewrites $m-1$ times A'' to bA'', and then rewrites A'' to bQ_3, $Prog_2$ always rewrites A' to A' and $Prog3$ rewrites D to D', then D' to D'' and rewrites $m-1$ times D'' to dD''

$\big\{(w_1 = a^n b^m Q_3) \wedge (w_2 = c^n a^n A') \wedge (w_3 = c^n d^{m-1} D'') \wedge (n, m \geq 1)\big\}$

Refinement for Subproblem 4 and Subproblem 5 is very simple:

$\big\{(w_1 = a^n b^m Q_3) \wedge (w_2 = c^n a^n A') \wedge (w_3 = c^n d^{m-1} D'') \wedge (n, m \geq 1)\big\}$

Subproblem 4: *Communication*

$\big\{(w_1 = a^n b^m c^n d^{m-1} D'') \wedge (w_2 = c^n a^n A') \wedge (w_3 = c^n d^{m-1} D'') \wedge (n, m \geq 1)\big\}$

Subproblem 5: *Rewrite*

$Prog_1$ rewrites D'' to d

$\{(w_1 \in \{a^n b^m c^n d^m) \wedge (n, m \geq 1)\}\}$

Equivalently we proposed a non-returning, non-centralized grammar system Γ with three regular components, defined in this way:

$$\Gamma = (N, K, \{a, b, c, d\}, (P_1, S_1), (P_2, S_2), (P_3, S_3))$$

where:

$N = \{S_1, S_2, S_3, A, A', A'', B, D, D', D''\}$

$K = \{Q_1, Q_2, Q_3\}$

$P_1 = \{S_1 \longrightarrow aS_1, S_1 \longrightarrow aA, A \longrightarrow A', A' \longrightarrow A'', A'' \longrightarrow bA'', A'' \longrightarrow bQ_3,$
$D'' \longrightarrow d\}$

$P_2 = \{S_2 \longrightarrow cS_2, S_2 \longrightarrow cB, B \longrightarrow Q_1, A' \longrightarrow A'\}$

$P_3 = \{S_3 \longrightarrow S_3, S_3 \longrightarrow Q_2, B \longrightarrow D, D \longrightarrow D', D' \longrightarrow D'', D'' \longrightarrow dD''\}.$

3. The last and hardest step is to prove global correctness. In our case it means that we have to show using Owicki-Gries theory that the multiprogram $\mathcal{P}$ we constructed satisfies this specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}\mathcal{P}\{w_1 \in L_{cd}\}.$$

Furthermore, $\mathcal{P}$ outputs the word $a^n b^m c^n d^n$ for any input formed by the pair of positive integers n, m. This is equivalent to prove that $L(\Gamma) = L_{cd}$.

According to how we defined $Prog_1$, $Prog_2$ and $Prog_3$, behaving like G_1, G_2 and G_3, respectively, we propose the following state:

$$\left\{ \begin{array}{c} \left[\begin{array}{c} (w_1 = a^n S_1 \wedge n \geq 0) \vee (w_1 = a^n A \wedge n \geq 1) \vee (w_1 = a^n A' \wedge n \geq 1) \vee \\ \vee (w_1 = a^v b^n A" \wedge v \geq 1 \wedge n \geq 0) \vee (w_1 = a^v b^n Q_3 \wedge v \geq 1 \wedge n \geq 1) \vee \\ \vee (w_1 = a^v b^n c^g d^h D" \wedge v, n, g \geq 1 \wedge h \geq 0) \vee (w_1 = a^e b^f c^g d^h \wedge e, f, g, h \geq 1) \end{array} \right] \wedge \\ \wedge \left[\begin{array}{c} (w_2 = c^q S_2 \wedge q \geq 0) \vee (w_2 = c^q B \wedge q \geq 1) \\ (w_2 = c^q Q_1 \wedge q \geq 1) \vee (w_2 = c^q a^r A' \wedge q, r \geq 1) \end{array} \right] \wedge \\ \wedge \left[\begin{array}{c} (w_3 = S_3) \vee (w_3 = Q_2) \vee (w_3 = c^n B \wedge n \geq 1) \vee (w_3 = c^n D \wedge n \geq 1) \vee \\ \vee (w_3 = c^n D' \wedge n \geq 1) \vee (w_3 = c^n d^m D" \wedge n \geq 1 \wedge m \geq 0) \end{array} \right] \end{array} \right\}$$

But by Owicki-Gries theory of global correctness one can prove that after n rewritings, with $n \geq 1$, the only possible combination of values for the sentential forms w_1, w_2 and w_3 that do not lead to a deadlock, is the one expressed by the state:

$$\{(w_1 = a^n A) \wedge (w_2 = c^n B) \wedge (w_3 = Q_2) \wedge (n \geq 1)\}$$

And from this state it can be proved that the only valid continuation is the sequence of rewritings and communications described in step 2 of the refinement process, that reaches to the state containing $\{w_1 \in \{a^n b^m c^n d^m \mid n, m \geq\}\}$ □

We have proved in the framework of programming that it is possible to generate L_{cd} with a grammar system with three components with regular productions, working in non-returning, non-centralized way. The strategy we have presented differs from the traditional approach not in complexity, because the number of cases considered in the proofs are the same, but in the way of reasoning about the problem. We state that Owicki-Gries methodology provides more possibilities for reasoning about problems, in comparison with the strategies used so far in grammar system framework because:

- it allows to reason in a forward or data-driven way, as analysis by cases techniques, but also in a backward or goal-directed way.

The notion of backward reasoning comes from psychology, as it is pointed in [9] where this description of problem solving occurs: *We may have a choice between starting with where we wish to end, or starting with where we are at the moment. In the first instance we start by analyzing the goal. We ask, "Suppose we did achieve the goal, how would things be different- what subproblems would we have solved, etc.?". This in turn would determine the sequence of problems, and we would work back to the beginning. In the second instance we start by analyzing the present situation, see the implications of the given conditions and lay-out, and attack the various subproblems in a "forward direction".*

- the division of problems in subproblems is possible because of the theorem: for any Q $\{P\}\mathcal{S}_0; \mathcal{S}_1\{R\} \Longleftarrow \{P\}\mathcal{S}_0\{Q\} \wedge \{Q\}\mathcal{S}_1\{R\}$, where P,R are predicates and $\mathcal{S}_0, \mathcal{S}_1$ are instructions. Also goals and subgoals are discussed in the psychology text mentioned above ([9]): *The person perceives in his surrounding goals capable of removing his needs and fulfilling his desires... And there is the important phenomenon of emergence of subgoals. The pathways to goals are often perceived as organized*

into a number of subparts, each of which constitutes and intermediate subgoal to be attained on the way to the ultimate goal.

These characteristics make Owicki-Gries strategies more related with human way of reasoning.

Now we want to prove a negative result about grammar systems: There is no grammar system of any type with two regular components that can generate L_{cd}. In other words, the solution we proposed is the most economical one with respect to the number of components. If we translate this problem into the programming framework, we have to prove that it is not possible to find a multiprogram $\mathcal{P}$ with two programs $\mathcal{P}_1$ and $\mathcal{P}_2$ running concurrently, modifying w_1 and w_2 in a right-linear way, that is correct with respect to this specification: $\{(w_1 = S_1) \wedge (w_2 = S_2)\}\mathcal{P}\{w_1 \in \{a^n b^m c^n d^m \mid n, m \geq\}\}$. But unfortunately we have no strategy or result in the concurrent programming theory that could help us to reasoning about this problem. The only strategies available in this framework are: *verification* that allows to prove the correctness of a multiprogram with respect to a specification, and the *constructive approach* that was exemplified in the previous proof. But these strategies are useful to get positive results therefore we have to remain in the grammar system framework to deal with this kind of problems. We solve it with the tools available there, namely analysis by cases.

Theorem 2 $L_{cd} \notin X_2(REG)$, *for* $X \in \{PC, CPC, NPC, NCPC\}$.

Proof. Since $PC_2(REG)$ (hence $CPC_2(REG)$, too), contains context-free languages only ([3]), it suffices to prove that L_{cd} cannot lie in $NPC_2(REG)$. Assume that $L_{cd} = L(\Gamma)$ for some non-returning non-centralized grammar system with two regular components Γ. Take $w = a^n b^m c^n d^m$ with arbitrarily large n, m. There exist two nonterminals A_1, A_2 such that in the process of generating w the following hold:

$$(S_1, S_2) \Longrightarrow^* (x_1 A_1, x_2 A_2), x_1, x_2 \in \{a, b, c, d\}^*,$$
$$(A_1, A_2) \Longrightarrow^{k_1} (uA_1, vA_2), u, v \in \{a, b, c, d\}^*, uv \neq \lambda$$

for some $k_1 \geq 1$. Here $\Longrightarrow^p$ denotes a derivation of length p where the communication steps are also counted. Let (A_1, A_2) be the first pair of such nonterminals met in the process with this property. By the choice of w we infer that $u \in a^+$ and $v \in c^+$. First we note that both u and v, if non-empty, are formed by one letter only. Second, if one is empty, then the other can be "pumped" arbitrarily many times which leads to a parasitic word. Third, by the same argument, all the other choices, except $u \in a^+$ and $v \in c^+$, lead to parasitic words. Since the subword of w formed by b is arbitrary long there is a pair of nonterminals (B_1, B_2) such that the derivation continues as follows:

$$(S_1, S_2) \Longrightarrow^* (x_1 A_1, x_2 A_2) \Longrightarrow^{rk_1} (x_1 u^r A_1, x_2 v^r A_2) \Longrightarrow^k$$
$$(x_1 u^r y_1 B_1, x_2 v^r y_2 B_2) \Longrightarrow^{pk_2} (x_1 u^r y_1 s^p B_1, x_2 v^r y_2 t^p B_2) \Longrightarrow^* (w, \alpha)$$

for some terminal words y_1, y_2, s, t, $s \in b^+$, positive integers r, p, k, k_2, and word α. Moreover, no communication step appears in the subderivation

$$(x_1 u^r A_1, x_2 v^r A_2) \Longrightarrow^* (x_1 u^r y_1 B_1, x_2 v^r y_2 B_2) \Longrightarrow^{pk_2} (x_1 u^r y_1 s^p B_1, x_2 v^r y_2 t^p B_2)$$

otherwise either an insufficient number of b's is generated between the two subwords formed by a and c or the generated word contains the subword cb which is contradictory.

Therefore, the following derivation is also possible in Γ:

$$(S_1, S_2) \Longrightarrow^* (x_1A_1, x_2A_2) \Longrightarrow^{rk_1} (x_1u^rA_1, x_2v^rA_2) \Longrightarrow^{k_1k_2+k}$$
$$(x_1u^{r+k_2}y_1B_1, x_2v^ry_2t^{k_1}B_2) \Longrightarrow^{pk_2} (x_1u^{r+k_2}y_1s^pB_1, x_2v^ry_2t^{p+k_1}B_2) \Longrightarrow^* (w', \alpha'),$$

for terminal word w' and word α'. However, w' cannot be in L_{cd}, which concludes the proof. □

We encourage the reader to translate this proof into the programming framework, where the reasoning is the same, but more explanations are needed.

4 Conclusions

The traditional approach to the problem of finding a grammar system generating a given language is: first propose a grammar system and then find a proof that it generates the language.

In this paper we present a new approach, taken from programming framework, that consists in finding simultaneously the grammar system that generates the given language and a proof that the grammar system found generates it. We think that it would be interesting to study this approach deeper trying to apply it to other well-known languages, or trying to find other programming strategies, apart from the strategy of refinement of problems shown here, that could be useful in solving some problems related to grammar system theory.

Until now main efforts in grammar system theory have been dedicated to find grammar systems with the smallest number of components or different types of restrictions applied to productions, to show how distribution and communication can make simple components be very powerful when they work together. So except for some studies related with computational complexity measure of PC grammar systems that considers the number of communication between grammars (see for example [8] and [13]), the most investigated complexity measure has been the number of grammars that a PC grammar system consist of, which is clearly a descriptional complexity measure. So a very important matter has been forgotten: the efficient use of time. The opposite has happened in the programming area (see [6] and [12]), where the research has been focused in looking for techniques to parallelize algorithms and to help programmers to design more efficient concurrent algorithms. We propose to follow some of the methodical approaches developed in the programming framework to construct more efficient grammar systems.

Also it would be very interesting not only to think how PC grammar system theory can benefit from concurrent programming, but how programming theory can benefit from grammar system theory. The lack of strategies in the programming framework to prove negative results of the type: $L \neq L(\Gamma)$ for a language L and any grammar system Γ, make us to think that such problems might be solved by translating them into the grammar system framework where they can be solved using the tools available there.

References

[1] A. Burns, G. Davies, *Concurrent Programming*, Addison-Wesley, Wokingham, England (1993).

[2] A. Chitu, PC grammar systems versus some non-context free constructions from natural and artificial languages, *New Trends in Formal Languages* (Gh. Păun, A. Salomaa, eds.), LNCS 1218, Springer-Verlag, Berlin (1997) 278-287.

[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] Dijkstra, W. Edsger, *A Discipline of Programming*, Prentice-Hall Series in Automatic Computation (1976).

[5] W. H. J. Feijen, A. J. M. van Gasteren (Eds.), *On A Method of Multi-Programming*, Springer-Verlag, Berlin (1999).

[6] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley (1995).

[7] D. Gries, *The Science of Programming*, Springer-Verlag, Berlin (1981).

[8] J. Hromkovič, J. Kari, L. Kari, Some hierarchies for the communication complexity measures of cooperating grammar systems, *Theoretical Computer Science* 127, 1 (1994) 123-147.

[9] D. Krech, R. S. Cruthfield, *Elements of Psycology*, Knopf, New York (1958) 383.

[10] C. Martin-Vide, V. Mitrana, Parallel communicating automata systems- a survey, *Korean J. Comput. Appl. Math.* 7, 2 (2000) 237-257.

[11] S. Owicki, D. Gries, An axiomatic proof technique for parallel programs 1, *Acta Informatica* 6 (1976) 319-340.

[12] D. Parnas, P. Clements, A rational design process: How and why to fake it. *IEEE Trans. Software Eng.* SE-12(2) (1986) 251–257.

[13] D. Pardubská, On the power of communication structure for distributive generation of languages, *Developments in Language Theory, At the Crossroads of Mathematics, Computer Science and Biology* (G. Rozenberg, A. Salomaa, eds.), World Scientific, Hugapore (1993) 419-429.

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

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

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 212 - 224, 2004.

DNA Involutions and Hairpin Structures

Lila Kari
Department of Computer Science, The University of Western Ontario,
London, ON, Canada, N6A 5B7,
lila@csd.uwo.ca

Stavros Konstantinidis
Dept. of Mathematics and Computing Science, Saint Mary's University,
Halifax, Nova Scotia, B3H 3C3 Canada,
s.konstantinidis@stmarys.ca

Elena Losseva
Department of Computer Science, The University of Western Ontario,
London, ON, Canada, N6A 5B7,
elena@csd.uwo.ca

Petr Sosík
Department of Computer Science, The University of Western Ontario,
London, ON, Canada, N6A 5B7,
sosik@csd.uwo.ca
and
Institute of Computer Science, Silesian University, Opava, Czech Republic

Gabriel Thierrin
Department of Mathematics, The University of Western Ontario,
London, ON, Canada, N6A 5B7,
thierrin@uwo.ca

Abstract

We formalize the notion of a DNA hairpin secondary structure, examining its mathematical properties. Two related secondary structures are also investigated, taking into account imperfect bonds (bulges, mismatches) and multiple hairpins. We characterize maximal sets of hairpin-forming DNA sequences, as well as hairpin-free ones. We study their algebraic properties and their computational complexity. Related polynomial-time algorithms deciding hairpin-freedom of regular sets are presented. Finally, effective methods for design of long hairpin-free DNA words are given.

1 Introduction

A single strand of deoxyribonucleic acid (DNA) consists of a sugar-phosphate backbone and a sequence of nucleotides attached to it. There are four types of nucleotides denoted by A, C, T, and G. Two single strands can bind to each other if they have opposite polarity (strand's orientation in space) and are pairwise Watson-Crick complementary: A is complementary to T, and C to G. The binding of two strands is also called annealing. The ability of DNA strands to anneal to each other allows for creation of different secondary structures. A DNA hairpin is a particular type of secondary structure investigated in this paper. An example of hairpin structure is shown in Figure 1.

```
                    C C
                   A     A
GTCAGCGATAG       A       T
CAGTCGCTATC       A C C   T
```

Figure 1: Hairpin loop.

The reader is referred to [9, 17] for an overview of the DNA computing paradigm. Study of DNA secondary structures such as hairpin loops is motivated by finding reliable encodings for DNA computing techniques. These techniques usually rely on a certain set of DNA bonds and secondary structures, while other types of bonds and structures are undesirable. Various approaches to the design of DNA encodings without undesirable bonds and secondary structures are summarized in [15]. For more details we refer the reader e.g. to [3, 4, 13, 14]. Here we apply the formal language approach which has been used in [1, 7, 10, 11, 12] and others.

Hairpin-like secondary structures are of special importance for DNA computing. For instance, they play an important role in insertion/deletion operations with DNA. Hairpins are the main tool used in the Whiplash PCR computing techniques [19]. In [21] hairpins serve as a binary information medium for DNA RAM. Last, but not least, hairpins are basic components of "smart drugs" [2].

The paper is organized as follows. Section 2 introduces basic definitions, Section 3 presents results on hairpins, and the following two sections define important variants on the hairpin definition. The first one takes into account imperfect DNA bonds (mismatches, bulges), the second one is related to hairpin-based nanomachines. We characterize maximal hairpin-free languages via hypercodes and study their algebraic properties. The hairpin-freedom problem and the problem of maximal hairpin-free sets are both shown to be decidable in linear (cubic) time for regular (context-free, respectively) DNA languages. The last section provides methods of constructing long hairpin-free words.

2 Preliminary Definitions

In this paper we will use X to denote a finite alphabet and X^* its corresponding free monoid. The cardinality of the alphabet X is denoted by $|X|$. The empty word

is denoted by 1, and $X^+ = X^* - \{1\}$. A *language* is an arbitrary subset of X^*. A *uniform (block) code* is a language all the words of which are of the same length k, for some $k \geq 0$.

A mapping $\psi : X^* \to X^*$ is called a *morphism* (*anti-morphism*) of X^* if $\psi(uv) = \psi(u)\psi(v)$ (respectively $\psi(uv) = \psi(v)\psi(u)$) for all $u, v \in X^*$, and $\phi(1) = 1$. See [5] for a general overview of morphisms. An involution $\theta : X \longrightarrow X$ is defined as a map such that θ^2 is the identity function. An involution θ can be extended to a morphism or an antimorphism over X^*. In both cases θ^2 is the identity over X^* and $\theta^{-1} = \theta$.

In our examples we shall refer to the DNA alphabet $\Delta = \{A, C, T, G\}$, on which several involutions of interest are defined. The simplest involution is the identity function ϵ. An antimorphic involution which maps each letter of the alphabet to itself is called a *mirror involution* and it is denoted by μ. The DNA *complementarity involution* γ is a morphism given by $\gamma(A) = T$, $\gamma(T) = A$, $\gamma(C) = G$, $\gamma(G) = C$. For example, $\epsilon(ACGCTG) = ACGCTG = \mu(GTCGCA) = \gamma(TGCGAC)$.

Finally, the antimorphic involution $\mu\gamma$ (the composite function of μ and γ, which is also equal to $\gamma\mu$), called the *Watson-Crick involution*, corresponds to the DNA bond formation of two single strands. If for two strings $u, v \in \Delta^*$ it is the case that $\mu\gamma(u) = v$, then the two DNA strands represented by u, v anneal as Watson-Crick complementary sequences.

A nondeterministic finite automaton (*NFA*) is a quintuple $A = (S, X, s_0, F, P)$, where S is the finite and nonempty set of states, s_0 is the start state, F is the set of final states, and P is the set of productions of the form $sx \to t$, for $s, t \in S$, $x \in X$. If for every two productions $sx_1 \to t_1$ and $sx_2 \to t_2$ of an NFA we have that $x_1 \neq x_2$ then the automaton is called a *DFA* (deterministic finite automaton). The language accepted by the automaton A is denoted by $L(A)$. The *size* $|A|$ of the automaton A is the number $|S| + |P|$.

Analogously we define a *pushdown automaton (PDA)* and a *deterministic pushdown automaton (DPDA)*. We refer to [6, 20] for detailed definitions and further information on formal language theory.

3 Hairpins

Definition 1 If θ is a morphic or antimorphic involution of X^* and k is a positive integer, then a word $u \in X^*$ is said to be θ-k-hairpin-free or simply hp(θ,k)-free if $u = xvy\theta(v)z$ for some $x, v, y, z \in X^*$ implies $|v| < k$.

Notice that the words 1 and $a \in X$ are hp(θ,1)-free. More generally, words of length less than $2k$ are hp(θ,k)-free. If we interpret this definition for the DNA alphabet Δ and the involution $\mu\gamma$, then a hairpin structure with the length of bond greater than or equal to k is a word that is not hp(θ,k)-free.

Definition 2 Denote by $hpf(\theta, k)$ the set of all hp(θ,k)-free words in X^*. The complement of $hpf(\theta, k)$ is $hp(\theta, k) = X^* - hpf(\theta, k)$.

Note that $hp(\theta, k)$ is the set of words in X^* which are hairpins of the form $xvy\theta(v)z$ where the length of v is at least k. In particular, if $w \in hp(\theta, k)$, then w can be

written as $w = xvy\theta(v)z$ with $|v| = |\theta(v)| = k$ for some $x, v, y, z \in X^*$. It is also the case that $hp(\theta, k+1) \subseteq hp(\theta, k)$ for all $k > 0$.

Definition 3 A language L is called θ-k-hairpin-free or simply hp(θ, k)-free if $L \subseteq hpf(\theta, k)$.

It is easy to see from the definition that a language L is hp(θ, k)-free *iff* $X^*vX^*\theta(v)X^* \cap L = \emptyset$ for all $|v| \geq k$.

Examples.

(1) Let $X = \{a, b\}$ with $\theta(a) = b, \theta(b) = a$. Then:

$$hpf(\theta, 1) = a^* \cup b^*$$

This example shows that in general the product of hp$(\theta, 1)$-free words is not a hp$(\theta, 1)$-free word. Indeed, a and b are hp$(\theta, 1)$-free, but the product ab is not.

(2) If $\theta = \gamma$ is the DNA complementary involution over Δ^*, then:

$$hpf(\theta, 1) = \{A, C\}^* \cup \{A, G\}^* \cup \{T, C\}^* \cup \{T, G\}^*$$

(3) Let $\theta = \mu$ be the mirror involution and let $u \in hpf(\theta, 1)$. Since $\theta(a) = a$ for all $a \in X$, u cannot contain two occurrences of the same letter a. This implies that $hpf(\theta, 1)$ is finite. For example, if $X = \{a, b\}$, then:

$$hpf(\theta, 1) = \{1, a, b, ab, ba\}$$

We now investigate properties of the languages $hp(\theta, k)$ and $hpf(\theta, k)$. Recall that $hp(\mu, k)$ is the set of all words containing two non-overlapping mirror parts of length at least k.

Lemma 1 *Consider a binary alphabet X. Then $hpf(\mu, k)$ is finite if and only if $k \leq 4$.*

Proof. Denote $X = \{a, b\}$ and consider a word w in the set $hpf(\mu, 4)$. Suppose that w is arbitrarily long. This word is of the form

$$w = a^{r_1} b^{s_1} \cdots a^{r_n} b^{s_n},$$

with $r_1 \geq 0$ and $r_i \geq 1$, for $i > 1$, and $s_n \geq 0$ and $s_j \geq 1$, for $j < n$. As w is in $hpf(\mu, 4)$, one has that $r_i < 8$ and $s_i < 8$, for all i, and n is arbitrarily large.

Consider now indices i_1, i_2, j_1, j_2 such that $1 < i_1, i_2, j_1, j_2 < n$. There can be at most one such index i_1 with $r_{i_1} \geq 3$ – otherwise, both ba^3 and a^3b would occur in w. By symmetry, there is at most one j_1 with $s_{j_1} \geq 3$. Moreover, if there is an index i_2 such that $r_{i_2} = 2$, then either there is no other index with this property, or it is the case that $s_{i_2} = 1$ and $r_{i_2+1} = 2$ and no other index i exists with $r_i = 2$. By symmetry again, there can be an index j_2 such that $s_{j_2} = 2$, and possibly $s_{j_2+1} = 2$, and no other index j exists with $s_j = 2$. Hence, we have that $r_k = 1$ for all k except i_1, i_2 and possibly 1, n and $i_2 + 1$, and $s_l = 1$ for all indices l except j_1, j_2

and possibly 1, n and $j_2 + 1$. As w is long enough, it would contain the substring $abababa$, which contradicts the assumption that $w \in hpf(\mu, 4)$. Hence, w cannot be arbitrarily long and, therefore, the set $hpf(\mu, 4)$ is finite.

Now consider the language $L_5 = (aabbab)^+$. The set of its subwords of length 5 is $Sub_5(L_5) = \{aabba, abbab, bbaba, babaa, abaab, baabb\}$. For its mirror image $\mu(L_5)$ we obtain $Sub_5(\mu(L_5)) = \{abbaa, babba, ababb, aabab, baaba, bbaab\}$. As these two sets are mutually disjoint, $L_5 \subseteq hpf(\mu, 5)$.

Finally, notice that for $1 \leq m \leq k$, finiteness of $hpf(\mu, k)$ implies also finiteness of $hpf(\mu, m)$. Hence the facts that $hpf(\mu, 4)$ is finite and $hpf(\mu, 5)$ is infinite imply the statement of the lemma. □

Proposition 1 *Let θ be a morphic or antimorphic involution. The language $hpf(\theta, k)$ over a non-singleton alphabet X is finite if and only if one of the following holds:*
(a) $\theta = \epsilon$, the identity involution;
(b) $\theta = \mu$, the mirror involution, and either $k = 1$ or $|X| = 2$ and $k \leq 4$.

Proof. (a) Let θ be a morphism. Assume first that $\theta \neq \epsilon$. Then there are $a, b \in X$, $a \neq b$, such that $\theta(a) = b$. Then $a^+ \subseteq hpf(\theta, k)$ for any $k \geq 1$, hence $hpf(\theta, k)$ is infinite.

Assume now that $\theta = \epsilon$ and let w be any word of length $\geq k|X|^k + k$. Since there are $|X|^k$ distinct words of length k, there are at least two subwords of length k in w which are identical. Hence $w = xvyvz$ for some $v \in X^k$ and $x, y, z \in X^*$. Therefore $hpf(\epsilon, k)$ is finite since it cannot contain any word longer than $k|X|^k + k$.

(b) Let θ be an anti-morphism. Assuming that $\theta \neq \mu$, the same arguments as above show that $hpf(\theta, k)$ is infinite.

Assume now that $\theta = \mu$. Apparently $hpf(\mu, 1)$ is finite as shown in the examples above. For $|X| = 2$ we know that $hpf(\mu, k)$ is finite *iff* $k \leq 4$ by Lemma 1. Finally, for $|X| > 2$ and $k > 1$ the language $hpf(\mu, k)$ is infinite as it always contains the hp$(\mu, 2)$-free set $(abc)^+$ (regardless to renaming the symbols).

□

3.1 Properties of hp$(\theta, 1)$-free languages

Recall the definition of an embedding order: $u \leq_e v$ if and only if

$$u = u_1 u_2 \cdots u_n, \ v = v_1 u_1 v_2 u_2 \cdots v_n u_n v_{n+1}$$

for some integer n with $u_i, v_j \in X^*$.

A language L is called *right $\leq_e$-convex* [22] if $u \leq_e w$, $u \in L$ implies $w \in L$. The following result is well known: *All languages (over a finite alphabet) that are right$\leq_e$-convex are regular.* We show that $hp(\theta, 1)$ is right $\leq_e$-convex (and hence regular).

Lemma 2 *If $u = u_1 u_2 \in hp(\theta, 1)$ and $w \in X^*$ then $u_1 w u_2 \in hp(\theta, 1)$.*

Proposition 2 *The language $hp(\theta, 1)$ is right $\leq_e$ -convex.*

Proof. Immediate. □

Let $L \subseteq X^*$ be a nonempty language and let:

$$S(L) = \{w \in X^* | u \leq_e w, u \in L\}.$$

Hence $S(L)$ is the set of all the words $w \in X^*$ that can be expressed in the form $w = x_1u_1x_2u_2 \cdots x_nu_nx_{n+1}$ with $u = u_1u_2 \cdots u_n \in L$ and $x_i \in X^*$.

Recall that a set H with $\emptyset \neq H \subseteq X^+$ is called a *hypercode* over X^* *iff* $x \leq_e y$ and $x, y \in H$ imply $x = y$. That is, a hypercode is an independent set with respect to the embedding order.

Proposition 3 *Let θ be a morphic or antimorphic involution. Then there exists a unique hypercode H such that $hp(\theta, 1) = S(H)$.*

Proof. Let $H = \bigcup_{a \in X} a\theta(a)$, then the result is immediate. □

Example. Consider the hypercodes for the earlier three examples.

1. For $X = \{a, b\}$ and the involution (morphic or antimorphic) $\theta(a) = b$, $\theta(b) = a$, the hypercode is $H = \{ab, ba\}$.
2. For the DNA complementarity involution γ we have $H = \{AT, TA, CG, GC\}$.
3. The mirror involution over $\{a, b\}^*$ gives the hypercode $H = \{aa, bb\}$.

3.2 Properties of hp(θ, k)-free languages

The previous results, Lemma 2 and Proposition 2, true for the case $k = 1$, cannot in general be extended to the case $k > 1$ as shown in the following example.

Example. Let $X = \{a, b\}$ with morphic $\theta(a) = b$, $\theta(b) = a$. If $u = a^2b^2$, then $u = a^2\theta(a^2)$ and hence $u \in hp(\theta, 2)$. The word $w = abab^2 = a.b.abb$ is obtained from u by the insertion of the word b and it is immediate that $w \notin hp(\theta, 2)$.

Furthermore, the language $hp(\theta, 2)$ is not $\leq_e$-convex, because $u \leq_e w$, $u \in hp(\theta, 2)$ and $w \notin hp(\theta, 2)$. However, the following result can be shown about $hpf(\theta, k)$-free languages, which indicates a possible construction method.

Proposition 4 *If $u = u_1u_2 \in hp(\theta, 2k)$ and $w \in X^+$, then $u_1wu_2 \in hp(\theta, k)$ for any $k \geq 1$.*

Proof. Let $u = u_1u_2 \in hp(\theta, 2k)$. Then u can be written as $u = xvy\theta(v)z$ with $|v| = |\theta(v)| = 2k$. The word u has v and $\theta(v)$ as substrings, which can be changed by the insertion of w into u only if insertion happens either in the middle of v or in the middle of $\theta(v)$. In the first case; $v_1wv_2, \theta(v_1v_2)$ are substrings of u_1wu_2 with $max(|v_1|, |v_2|) \geq k$, hence $u_1wu_2 \in hp(\theta, k)$. The second case is similar. □

Proposition 5 *The languages $hp(\theta, k)$ and $hpf(\theta, k)$, $k \geq 1$, are regular.*

Proof. The language $hp(\theta, k)$ can be written as $hp(\theta, k) = \bigcup_{|w| \geq k} X^* w X^* \theta(w) X^*$. This language is furthermore equal to $L = \bigcup_{|w|=k} X^* w X^* \theta(w) X^*$. Every language $X^* w X^* \theta(w) X^*$with $|w| = k$ is regular, hence L is a union of a finite number of regular languages. Therefore both $hp(\theta, k)$ and its complement $hpf(\theta, k)$ are regular. □

Corollary 1 *Let θ be a morphic or antimorphic involution and $k \geq 1$. The following problem of* hairpin-freedom *is decidable in linear time w.r.t. $|M|$:*

Input: *An NFA M.*

Output: *Yes/No depending on whether $L(M)$ is $hp(\theta, k)$-free.*

Proof. By definition, $L(M)$ is hp(θ, k)-free *iff* $L(M) \subseteq hpf(\theta, k)$ *iff* $L(M) \cap hp(\theta, k) = \emptyset$. This problem is solvable in linear time for regular languages. □

Also the *maximal hairpin-freedom* problem is highly relevant: given a hairpin-free language L_1 and a pool of DNA words L_2, is it possible to add new words from L_2 to L_1 without breaking its hairpin-freedom?

Corollary 2 *Let θ be a morphic or antimorphic involution and $k \geq 1$. The following problem is decidable in linear time w.r.t. $|M_1| \cdot |M_2|$:*

Input: *A DFA M_1 such that $L(M_1)$ is $hp(\theta, k)$-free, and a NFA M_2.*

Output: *Yes/No depending on whether there is a word $w \in L(M_2) - L(M_1)$ such that $L(M_1) \cup \{w\}$ is $hp(\theta, k)$-free.*

Proof. We want to determine if there exists a word $w \in hpf(\theta, k)$ such that $w \notin L(M_1)$, but $w \in L(M_2)$. It is decidable in time $\mathcal{O}(|M_1| \cdot |M_2|)$ whether $(hpf(\theta, k) \cap L(M_2)) - L(M_1) = \emptyset$. The size of an automaton accepting $hpf(\theta, k)$ is fixed for a chosen k. □

As an immediate consequence, for a given block code K of length l it is decidable in linear time with respect to $|K| * l$, whether there is a word $w \in X^l - K$ such that $K \cup \{w\}$ is hp(θ, k)-free. This is of particular interest since the lab sets of DNA molecules adopt often the form of a block code.

The above results can be extended also for the case of context-free languages.

Corollary 3 *Let θ be a morphic or antimorphic involution and $k \geq 1$. The following problem is decidable in cubic time w.r.t. $|M_1| \cdot |M_2|$:*

Input: *A DPDA M_1 such that $L(M_1)$ is $hp(\theta, k)$-free, and a NFA M_2.*

Output: *Yes/No depending on whether there is a word $w \in L(M_2) - L(M_1)$ such that $L(M_1) \cup \{w\}$ is $hp(\theta, k)$-free.*

Proof. We want to determine if $\exists w \in hpf(\theta, k)$ such that $w \notin L(M_1)$, but $w \in L(M_2)$. Denote $M_1 = (Q_1, X, \Gamma, q_1, Z_0, A_1, \delta_1)$, and let $M_2' = (Q_2, X, q_2, A_2, \delta_2)$ be a NFA accepting the language $hpf(\theta, k) \cap L(M_2)$. Consider the PDA $M = (Q, X, \Gamma, q_0, Z_0, A, \delta)$, where $Q = Q_1 \times Q_2$, $q_0 = (q_1, q_2)$. For $p \in Q_1, q \in Q_2$, and $Z \in \Gamma$ we define:

(1) $\delta((p,q), a, Z) = \{((p', q'), \alpha) | (p', \alpha) \in \delta_1(p, a, Z)$ and $\delta_2(q, a) = q'\}$

(2) $\delta((p,q), \lambda, Z) = \{((p', q), \alpha) | (p', \alpha) \in \delta_1(p, \lambda, Z)\}$

$A = \{(p,q) | p \notin A_1$ and $q \in A_2\}$ Then $L(M) = (hpf(\theta, k) \cap L(M_2)) - L(M_1)$, and the size of M is $\mathcal{O}(|M_1| \cdot |M_2|)$. Let G be a CFG such that $L(G) = L(M)$. Note that the construction of G takes cubic time w.r.t. $|M|$ (see Theorem 7.31 of [6]). Finally, it is possible to decide in linear time w.r.t. $|G|$ (see Section 7.4.3 of [6]) whether $L(G) = \emptyset$ or not. □

We can similarly extend the hairpin-freedom result in Corollary 1. In fact, the result is true for PDAs in general, including nondeterministic ones. The proof is remained to the reader.

Corollary 4 *Let θ be a fixed morphic or antimorphic involution and $k \geq 1$. The following problem is decidable in cubic time w.r.t. $|M|$:*

Input: *A PDA M.*

Output: *Yes/No depending on whether $L(M)$ is $hp(\theta, k)$-free.*

4 Scattered Hairpins

It is a known fact that parts of two DNA molecules could form a stable bond even if they are not exact mutual Watson-Crick complements. They may contain some mismatches and even may have different lengths. Hybridizations of this type are addressed e.g. in [1] and [12]. Motivated by this observation, we consider now a generalization of hairpins.

Definition 4 Let θ be an involution of X^* and let k be a positive integer. A word $u = wy \in X^*$ is θ-k-scattered-hairpin-free or simply shp(θ, k)-free if $t \leq_e w$, $\theta(t) \leq_e y$ implies $|t| < k$.

The set of words which are not shp(θ, k)-free characterizes DNA structures such as shown on Fig. 2. The depicted hairpin incorporates a bulge which is caused by a mismatched nucleotides within a double helix.

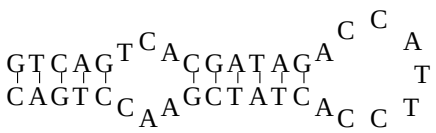

Figure 2: Bulge.

Definition 5 We denote by $shpf(\theta, k)$ the set of all shp(θ, k)-free words in X^*, and by $shp(\theta, k)$ its complement $X^* - shpf(\theta, k)$.

Example. Any word in $\{A, C\}^*$ is shp(θ, k)-free for every $k \geq 1$.

Definition 6 A language L is called θ-k-scattered-hairpin-free or simply shp(θ,k)-free if $L \subseteq shpf(\theta, k)$.

Lemma 3 $shp(\theta, k) = S\Big(\bigcup_{w \in X^k} w\theta(w)\Big)$.

Based on the above immediate result, analogous statements as in Section 3.1 hold also for scattered hairpins. Proofs are straightforward and left to the reader.

Proposition 6

(i) The language $shp(\theta, k)$ is right $\leq_e$ -convex.

(ii) The languages $shp(\theta, k)$ and $shpf(\theta, k)$ are regular.

(iii) There exists a unique hypercode H such that $shp(\theta, k) = S(H)$.

Corollary 5

(i) The scattered-hairpin-freedom *problem is decidable in linear time for regular languages and in cubic time for context-free languages.*

(ii) The maximal scattered-hairpin-freedom *problem is decidable in linear time for regular languages and in cubic time for deterministic context-free languages.*

5 Hairpin Frames

In this section we point out the following two facts. First, long DNA and RNA molecules can form complicated secondary structures as that shown in Figure 3. Second, simple hairpins are not always undesirable from the point of view of DNA computing. There exist nanomachines using simple hairpins, see e.g. [21]. Hairpins play also important role in some DNA computing techniques as we mentioned in Section 1. Hence it may be desirable to design DNA strands forming simple hairpins but avoiding more complex structures. Therefore we consider another extension of the results from Section 3.1.

Definition 7 The pair $(v, \theta(v))$ of a word u in the form $u = xvy\theta(v)z$, for $x, v, y, z \in X^*$, is called a *hp-pair* of u.

The sequence of hp-pairs $(v_1, \theta(v_1)), (v_2, \theta(v_2)), \cdots, (v_j, \theta(v_j))$ of the word u in the form:

$$u = x_1v_1y_1\theta(v_1)z_1x_2v_2y_2\theta(v_2)z_2 \cdots x_jv_jy_j\theta(v_j)z_j$$

is called a *hp-frame of degree j* of u or simply a *hp(j)-frame* of u.

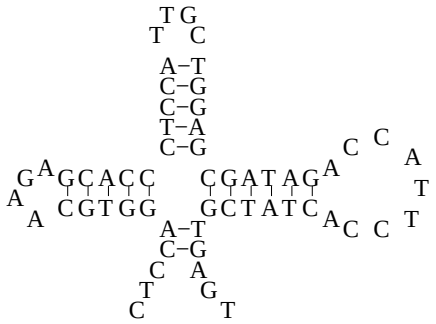

Figure 3: Secondary structure of a DNA strand with a hairpin frame.

An hp-pair is an hp-frame of degree 1. The definition of hairpin frames characterizes secondary structures containing several complementary sequences such as that in Fig. 3.

A word u is said a *hp(fr,j)-word* if it contains at least one hp-frame of degree j. Observe that there may be more ways how to find hp-pairs in u, resulting in hp-frames of various degrees. Obviously, any hp(fr,j)-word is also hp(fr,i) for all $1 \leq i \leq j$.

Definition 8 For an involution θ we denote by $hp(\theta, fr, j)$ the set of all hp(fr,j)-words $u \in X^*$, and by $hpf(\theta, fr, j)$ its complement in X^*.

Example. The word in Fig. 3 is in $hp(\theta, fr, 3)$, where $\theta = \mu\gamma$.

The results in Section 3, concerning the languages $hp(\theta, 1)$ and $hpf(\theta, 1)$, can easily be extended for the case of hairpin frames. Proofs are left to the reader.

Lemma 4 $hp(\theta, fr, j) = hp(\theta, 1)^j = \left(\bigcup_{a \in X} X^* a X^* \theta(a) X^* \right)^j$.

Proposition 7

(i) The language $hp(\theta, fr, j)$ is right $\leq_e$ -convex.

(ii) The languages $hp(\theta, fr, j)$ and $hpf(\theta, fr, j)$ are regular.

(iii) There exists a unique hypercode H such that $hp(\theta, fr, j) = S(H)$.

Corollary 6

(i) The hp(fr,j)-freedom *problem is decidable in linear time for regular languages and in cubic time for context-free languages.*

(ii) The maximal hp(fr,j)-freedom *problem is decidable in linear time for regular languages and in cubic time for deterministic context-free languages.*

6 Construction of Long Hairpin-Free Words

The following results hold for the case $\theta = \epsilon$. Let $H(K)$ denote the minimum Hamming distance between any two different codewords of a code K. A language K is said to be a solid code [8] if (i) no word of K is a subword of another word of K, and (ii) a proper and nonempty prefix of K cannot be a suffix of K. See [8, 18] for background information on codes.

Proposition 8 *Let $k \geq 2$ and let K be a uniform solid code of length k. If $H(K) > \lfloor k/2 \rfloor$, or $H(K) = \lfloor k/2 \rfloor$ and there are no different codewords with a common prefix of length $\lfloor k/2 \rfloor$, then the word $w_1...w_n$ is hp(θ, k)-free for all $n \leq \mathrm{card}(K)$ and for all pairwise different codewords $w_1, ..., w_n$.*

Proof. Assume there is $v \in X^k$ such that $w_1...w_n = xvyvz$ for some words x, y, z. If $|x|$ is a multiple of k then $v = w_j$ for some $j \geq 1$. As the w_i's are different, $|y|$ cannot be a multiple of k. Hence, $v = s_t p_{t+1}$, where $t > j$ and s_t is a proper and nonempty suffix of w_t and p_{t+1} is a proper and nonempty prefix of w_{t+1}; a contradiction. Now suppose $|x|$ is not a multiple of k. Then, $v = s_j p_{j+1}$ for some nonempty suffix s_j and prefix p_{j+1}. Again, the second occurrence of v cannot be in K. Hence, $v = s_t p_{t+1}$ for some $t \geq j$. Hence, $s_j p_{j+1} = s_t p_{t+1}$. If $|s_j| \neq |s_t|$, say $|s_j| > |s_t|$, then a prefix of p_{t+1} is also a suffix of s_j; which is impossible. Hence, $s_j = s_t$ and $p_{j+1} = p_{t+1}$.

Note that $H(K) \geq \lfloor k/2 \rfloor$ and, therefore, $\lfloor k/2 \rfloor \leq H(p_{j+1}s_{j+1}, p_{t+1}s_{t+1}) = H(s_{j+1}, s_{t+1}) \leq |s_{j+1}| = k - |p_{j+1}|$. Hence, $|p_{j+1}| \leq \lceil k/2 \rceil$. Similarly, $|s_j| \leq \lceil k/2 \rceil$. Also, as $k = |s_j| + |p_{j+1}|$, one has that $|s_j|, |p_{j+1}| \in \{\lfloor k/2 \rfloor, \lceil k/2 \rceil\}$. If $H(K) = \lfloor k/2 \rfloor$ then $p_{j+1} = p_{t+1}$ implies that w_{j+1} and w_{t+1} have a common prefix of length $\lfloor k/2 \rfloor$; a contradiction. If $H(K) > \lfloor k/2 \rfloor$ then both p_{j+1} and s_j are shorter than $\lceil k/2 \rceil$ which contradicts with $k = |s_j| + |p_{j+1}|$. □

Suppose the alphabet size $|X|$ is $l > 2$. We can choose any symbol $a \in X$ and consider the alphabet $X_1 = X - \{a\}$. Then for any uniform code $F \subseteq X_1^{k-1}$ it follows that the code Fa is a uniform solid code of length $k : Fa \subseteq X^k$. We are interested in cases where the code F is a linear code of type $[k-1, m, d]$. That is, F is of length $k-1$, cardinality $(l-1)^m$, and $H(F) = d$, and there is an m by $k-1-m$ matrix G over X_1 such that $F = \{w * [I_m|G] : w \in X_1^m\}$, where I_m is the identity m by m matrix and $*$ is the multiplication operation between a 1 by m vector and an m by m matrix. Thus, $u \in F$ *iff* $u = wx$ for some $w \in X_1^m$ and $x \in X_1^{k-1-m}$ and $x = wG$.

Proposition 9 *Let F be a linear code over X_1 of type $[k-1, m, \lfloor k/2 \rfloor]$. If $m \leq \lfloor k/2 \rfloor$ or k is even then the word $w_1..w_n$ is hp(θ, k)-free for all $n \leq \mathrm{card}(F)$ and for all pairwise different codewords $w_1, ..., w_n$ in Fa.*

Proof. It is sufficient to show that $H(Fa) = \lfloor k/2 \rfloor$ and there are no different words in Fa with a common prefix of length $\lfloor k/2 \rfloor$. Obviously $H(Fa) = H(F) = \lfloor k/2 \rfloor$. As F is generated by a matrix $[I_m|G]$, where G is a matrix in $X_1^{m \times (k-1-m)}$, it follows that there can be no different words in F with a common prefix of length m. If $m \leq \lfloor k/2 \rfloor$ then there can be no different words in Fa with a common prefix of length $\lfloor k/2 \rfloor$.

If k is even, consider the well known bound on $|F|$: $|F| \leq |X_1|^{k-1-\lfloor k/2 \rfloor+1}$. Hence, $|X_1|^m \leq |X_1|^{\lfloor k/2 \rfloor}$ which gives $m \leq \lfloor k/2 \rfloor$. Hence, again, we are done. $\square$

By the above one can construct an $\mathrm{hp}(\theta, k)$-free word of length nk, for some $n \leq \mathrm{card}(F)$, for every choice of n different words in Fa. It is interesting that, for $k = 13$ and $|X| = 4$, the famous Golay code G_{12} of type $[12, 6, 6]$ satisfies the premises of the above Proposition.

Acknowledgements

Research was partially supported by the Canada Research Chair Grant to L.K., NSERC Discovery Grants R2824A01 to L.K. and R220259 to S.K., and by the Grant Agency of Czech Republic, Grant 201/02/P079 to P.S.

References

[1] M. Andronescu, D. Dees, L. Slaybaugh, Y. Zhao, A. Condon, B. Cohen, S. Skiena. Algorithms for testing that sets of DNA words concatenate without secondary structure. In *Proc. 8th Workshop on DNA-Based Computers*, M. Hagiya, A. Ohuchi, Eds., *LNCS* 2568 (2002), 182–195.

[2] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature* 429 (2004), 423-429.

[3] R. Deaton, R. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens. Good encodings for DNA-based solutions to combinatorial problems. *DNA-based computers II*, in AMS DIMACS Series, vol.44, L.F.Landweber, E.Baum Eds., 1998, 247–258.

[4] J. Chen, R. Deaton, M. Garzon, J.W. Kim, D. Wood, H. Bi, D. Carpenter, Y.-Z. Wang. Characterization of non-crosshybridizing DNA oligonucleotides manufactured *in vitro*. In [16], 132–141.

[5] T. Harju, J. Karhumäki. Morphisms. In [20], 439–510.

[6] J. Hopcroft, J. Ullman, R. Motwani. Introduction to Automata Theory, Languages, and Computation, 2nd ed., Addison-Wesley, 2001.

[7] N. Jonoska, K. Mahalingam. Languages of DNA based code words. In *DNA Computing, 9th International Workshop on DNA Based Computers*, J. Chen and J.H. Reif, Eds., *LNCS* 2943 (2004), 61–73.

[8] H. Jürgensen, S. Konstantinidis. Codes. In [20], 511–607.

[9] L. Kari. DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer*, vol.19, nr.2, Spring 1997, 9–22.

[10] L. Kari, S. Konstantinidis, E. Losseva, G. Wozniak. Sticky-free and overhang-free DNA languages. *Acta Informatica* 40, 2003, 119–157.

[11] L. Kari, S. Konstantinidis, P. Sosík. Preventing undesirable bonds between DNA codewords. In [16], 375–384.

[12] L. Kari, S. Konstantinidis, P. Sosík. Bond-free languages: formalizations, maximality and construction methods. In [16], 16–25.

[13] S. Kobayashi. Testing Structure Freeness of Regular Sets of Biomolecular Sequences. In [16], 395–404.

[14] A. Marathe, A. Condon, R. Corn. On combinatorial DNA word design. *DNA based Computers V*, DIMACS Series, E.Winfree, D.Gifford Eds., AMS Press, 2000, 75–89.

[15] G. Mauri, C. Ferretti. Word Design for Molecular Computing: A Survey. In *DNA Computing, 9th International Workshop on DNA Based Computers*, J. Chen and J.H. Reif, Eds., LNCS 2943 (2004), 37–46.

[16] G. Mauri, C. Ferretti., Eds., *DNA 10, Tenth International Meeting on DNA Computing.* Preliminary proceedings, University of Milano-Bicocca, 2004.

[17] G. Paun, G. Rozenberg, A. Salomaa. *DNA Computing: New Computing Paradigms*, Springer Verlag, Berlin, 1998.

[18] S. Roman. *Coding and Information Theory,* Springer-Verlag, New York, 1992.

[19] J. A. Rose, R. J. Deaton, M. Hagiya, A. Suyama. PNA-mediated Whiplash PCR. In *DNA Computing, 7th International Workshop on DNA Based Computers*, N. Jonoska and N. C. Seeman, Eds., LNCS 2340 (2002), 104–116.

[20] G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, vol. 1, Springer Verlag, Berlin, 1997.

[21] N. Takahashi, A. Kameda, M. Yamamoto, A. Ohuchi, Aqueous computing with DNA hairpin-based RAM. In [16], 50–59.

[22] G. Thierrin. Convex languages. *Proc. IRIA Symp. North Holland* 1972, 481–492.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil, MTA SZTAKI, Budapest, pages 225 - 239, 2004.

On Computational Power of Interacting Bodies[1]

Jozef Kelemen
Institute of Computer Science, Silesian University
Bezruč Square 13, 746 01 Opava, Czech Republic
and Gratex International, a. s., Bratislava, Slovakia
kelemen@fpf.slu.cz

Abstract: After sketching the difference between implementation and embodiment, and some related problems concerning the Turing-type computation and the so-called hyper-computation, the contribution proposes EG-systems as a suitable computationalistic formal framework for study some of the relevant properties of embodied autonomous agents acting in computationally complicated environments.

1. Introduction

We have become very good at modeling fluids, materials, planetary dynamics, nuclear explosions and all manner of physical systems. Put some parameters into the program, let it crank, and out come accurate predictions of the physical character of modeled system. But we are not good at modeling living systems, at small or large scales. Something is wrong. What is wrong? There are a number of possibilities: (1) we might just be getting a few parameters wrong; (2) we might be building models that are below some complexity threshold; (3) perhaps it is still a lack of computing power; and (4) we might be missing something fundamental and currently unimaginable in our models wrote R. Brooks (2001, p. 401). The situation is similar studying cognition, intelligence, perception, etc. We use the traditional conceptual framework of studying computing devices and their behaviors interpreted as computation to dissolve the miracle of the mentioned phenomena. Having at hand the prepared set of notions and scientific rules which express possible relations between them we try to explain the nature of these phenomena. We are in certain extent successful in doing that. But fundamental difficulties even with formulation of some questions concerning these miraculous phenomena remain open, but there have high actuality for better understanding of the just arising completely new meaning of the concept of machines; for more details concerning the cultural and scientific evolution of the concept of machine during the 20th Century see (Horáková, Kelemen, 2003).

[1] The author's research on the subject is supported by the Grant Agency of the Czech Republic, grant No. 201/04/0528.

The concept of machine is not the only one which has been changed considerably during the second half of the 20th century. Important changes have been started already at the end of the 19th century in viewing the relation of human psyche and body thanks to the pioneering work of Sigmund Freud. In his unpublished during his life paper written in 1895 (Freud, 1954) he sketched the figure like the following one:

Wee can clearly recognize Freud's division of the psychic and the somatic (in the picture the horizontal *somat.-psych. Grenze*) as well as the humans "I" from the outer environment (in the picture the vertical *Ich-granze*) in his representation of the sexual function; cf. e.g. (Panhuysen, 1998). The continuous interaction of the human body with its environment is the base for complicated processes the psychic processes inclusive, which results in a specific state of the human mind and in performing some sexual behavior. Without the body as certain kind of "interface" the state of mind nor the related with it behavior does not emerge in human beings. From this fact, among other, follows the importance of the human body for human mind and bel

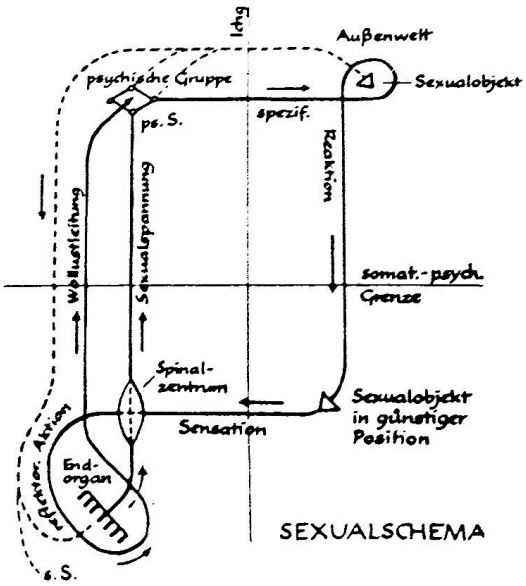

Fig. 1: Freud's schematic representation of the sexual function; from (Penhuysen, 1998).

Situation is very similar in nowadays research in many disciplines focused to human beings, to machines, or to the intersection of the both categories in some branches of science, cf. e.g. (Humphrey, 2000). Some of the specialists, esp. some of those working in the fields of cognitive science, artificial intelligence, and advanced robotics, argue that the source of problems with discovering more adequate and effective ways how construct (esp. how to program) machines in

order to provide their continuous functioning in dynamically changing environments consists in the embodiment of systems, the phenomenon which remained almost completely ignored in our recent computationalistic models. The traditional so called *mind-body problem* of philosophers and cognitive scientists, and the actual *software-hardware problem* of computer programmers and robot builders are form the perspective of embodiment in certain sense identical.

The core of the problem consists, according (Scheutz, 2002), in the simple fact that since symbols are abstract entities, computations cannot be performed on them, but have to be mediated through something physical (like organic bodies of living beings or inorganic machines) that can be manipulated by some physical operations that correspond systematically to the ones performed during the abstract computational processes over abstract symbols. Moreover, because of the same reasons the symbols themselves must be represented in certain ways by suitable physical entities. These entities are then manipulated by the above mentioned physical processes and the results of manipulation are reinterpreted as the results of an abstract computation. The just described *abstract-physical dichotomy* remains unmentioned at all. But when we concentrate to build embodied systems acting in a dynamic, often unpredictable environment, we are confronted with the question how the abstract and the physical is related and how this relation influenced the behavior of our robots, for instance. This is the core of the *problem of embodiment* (at least for the purposes of this paper).

The problem of embodiment is highly actual e.g. because of effective construction of different physically embodied autonomous agents but, unfortunately, we have no effective tools at hand to study them with theoretical rigor up to now. The aim of this contribution is to sketch, one the base of a short analysis of the story of Artificial Intelligence efforts and on some developments in of theoretical computer science, a computationally relevant sub-problem of the general problem of embodiment, and to sketch some of consequences of the previous developments for a way how to deal with this sub-problem in the frame of EG system.

2. The Very Short Story of Artificial Intelligence

An important achievement of Artificial Intelligence (AI) was the discovery of the methodologically new possibility how to test our hypotheses on how (some of) the intellectual processes run. The history of AI is full of different hypotheses on how to “automatize” processes like general problem solving, theorem proving, natural language understanding and communication, diagnostics, image processing and recognition, scene analysis, etc. in order to obtain working computer-based systems performing these tasks at the similar (or at better) qualitative level as (specially trained) human beings perform them. In all these cases:

(1) A working hypothesis is produced first – in the majority of the cases it is based on author's own introspection, then
(2) the formulated hypothesis is implemented (often using a suitable programming language that might be developed for such purposes), and
(3) the developed system of programs (the implemented version of the hypothesis) is then tested on real (or more or less similar to the real ones) data.

To proceed according such methodological guidance seems to us as something natural. It might be because intellectually we feel prepared for contemplations about our own intellectual capacities. Perhaps the most deeply developed system of this kind is the well-known system GPS (General Problem Solver) by A. Newell, H. A. Simon and their collaborators (Newell, Simon, 1972), (Ernst, Newell, 1969) continued in the frame of the project SOAR (Newell, 1990).

The most illustrative achievements of the use of the above-sketched methodology are the knowledge systems [15] having symbolically represented ontologies of notions, their chunks, taxonomies, relations between them, etc. As the consequence of that, knowledge systems do not need any bodies (in the physical sense). The situation is completely different in the cases when the artificially created systems (intended to be intelligent in certain sense, e.g. cognitive robots) are situated and execute tasks in real physical environments. In such a case the systems are faced with physically grounded ontologies of objects with real physical properties that exist and act in real time scales. Very hard problems appearing in such situations in the traditional good old fashioned AI were pointed out firs from very different positions and with very different conclusions by M. Minsky (1986) and R. Brooks (1999).

Brooks (1999) in his concept of the new AI emphasizes the principal role of systems reactivity, which is necessary for their low-level rationality, while Minsky (1986) emphasizes the principle of decentralization and organization of simplest units (agents) into more complex ones (agencies) and presupposes that an agency may play the role of an agent in a more complex agency. Both of these positions might be – according to our conviction – combined into one unified approach. The main idea consists in two basic steps:

(1) in emphasizing the role of as direct as possible interaction of the cognitive systems with their environments at least at the lowest level of sensing and acting, and
(2) in exploiting the power of organization and of the emergence in highest levels in order to receive more complex behaviors.

Both of the above mentioned steps lead us to realize the principal difference between *implementation* of our ideas on how cognitive processes run in natural systems and how they may run in artificial ones, into more or less traditional but in certain sense rigid computers usually equipped with suitable input-output

devices which isolate them form their environments by providing data from it for them, and between *embodiment* of our ideas into artificially created systems equipped by sensors providing signals for them, by units for processing signals and perhaps compute the decisions, and by actuators for making changes in their environments, and situated and working continuously in real, dynamic, and noisy environments; for more details see e.g. (Ačová et al., 2004).

The bodies of our more or less smart machines became the principal problem of our scientific consideration. We have very deep experience with understanding physical machines as physical systems, e.g. in mechanics. However, as we have mentioned above, the mechanistic view of bodies is not sufficient when we are interested in behavioral aspects of functioning of machines. In computer science we are interested rather in virtual machines, in machines in the case of which we make a shift in abstracting behavioral aspects of these machines (the software) and exclude from any considerations their bodies (the hardware). This type of separation has been and still is fruitful in certain situation appearing when the computers are used in traditional ways, but is not sufficient in some other cases.

3. The Very Short Story of Traditional Computing

According this traditional understanding of computation and computers we can recognize any computing device as an *externally passive entity* which internal activity is based on the activities of some finite number of externally passive components with predefined message passing possibilities. Thank to the internal activities of these components and their addressed communication possibilities the whole system transforms some inputs provided to it from certain environment into some required outputs. If some well-specified requirements are satisfied, such transformation is a computation in the traditional sense developed during the modern history of computing and computation which started in 30ties of the past century with definition and first studies of (abstract devices equivalent with) the Turing machine.

Important from the traditional point of view is that considering a Turing machine working in an environment, it gets its input in advance at a beginning of it work, and outputs the result to the environment at the end of its activity. During the computation, the environment is – from the perspective of the Turing machine – completely passive. Computing and computation are understood, applying this traditional paradigmatic view to the systems understood as computing devices, as specific processes corresponding to mathematically defined functions. While the function *declares* a specific relation between variables and values in a set theoretic sense (to definition of a function coincides with a defining a suitable subset of the Cartesian product of its domain of variables and domain of its values), the traditional view of a computation (of a function) is *procedural* one: a computation define a function by means of specifying a step-by-step process of

elementary computable transformation steps which transform the given input variable to a corresponding output value (of the corresponding function).

The central problems of (theoretical) computer science originated from the point of view of the just described traditional paradigm of computing are related with the possibility, description, execution, and the effectiveness of an idealized rule governed *algorithmic* transformation of input data into the desired outputs.

Inside the above sketched overall picture of the traditional understanding of computation, the property of computability – or in other words the (partial) recursiveness (of mathematically defined functions) – is derived from the computing power of the Turing machine. This is the core idea of the so-called Church-Turing thesis, which, in a more precise formulation, states Turing machines, logics, lambda calculus, algorithmic computing, and the generative capacity of centralized rule-based systems (Chomsky-type formal grammars) as equivalent mechanisms for solving problems; cf. (Wegner, Goldin, 2003).

4. Hyper-Computation and Interacting Embodied Agents

However, in present there are strong efforts to prove that the notion of computation might be enlarged beyond the traditional boundaries defined by Turing computability[2]. In (Burgin, Klinger, 2004) it is proposed to call algorithms and automata that are more powerful than Turing machines as *super-recursive*, and computations that cannot be realized or simulated by Turing machines as *hyper-computations*. In our following consideration on the possible views of computation we will respect this proposal.

Another possibility of viewing systems as computing devices consists in considering a computing device as an *externally active entity* perceiving its dynamic (might be hardly predictable, noisy, or completely unpredictable) outer environment, and acting in it continuously according the perceived stimuli and the own inner rules governing the behavior of the system in order to complete given tasks. This is the core idea of the third period of the history of modern computing when the more or less freely cooperating and communicating interacting processors individual behaviors result in a behavior interpretable as a solution of a given problem. The interactivity, as stated in (Copeland, 2004) in connection with the analysis of the computational power of the Turing machine *coupled* with its environment, or with the same device appearing as the *interactive* Turing machine in (Wegner, 1997) leads to the hyper-computational power of the interacting in the Turing sense computationally universal devices.

[2] For more details on the effort see e.g. (Eberbach, Wegner, 2003) or the monothematic issue of *the Theoretical Computer Science* **317**, No. 1-3 (2004) 1-269.

The activity of the above mentioned type of systems is based on their own coupling of sensed data with appropriate acts performed in their environment, or on the activities of individually autonomous components forming these systems, and communicating (directly or indirectly) with other components forming them. Systems of this type are usually called *agents*, and the structures formed by these agents are called *multi-agent systems*; cf. e.g. (Ferber, 1999). In (Kelemen, 2003) we called the emerging new paradigm of considering computing systems are emerging from considering such kind of autonomous "open" systems as computing devices instead of the isolated ones as the *agent paradigm* of computing.

Interactions of agents with other agents and with their (dynamically changing, unpredictable, noisy, etc.) external environment during their activities in that environment are a real promise how to enlarge computational power of agents and multi-agent systems in comparison with the computational power of the Turing machine; cf. e.g. (Wagner, 1997). In general, interactions inside a multi-agent system involve the external word and the activities of individual agents into the behavior (interpreted as a computation) of the whole system *during* the computation (rather than *before* and *after*, as it is in the case of the traditional algorithms) which may lead to the computations that cannot be carried out by a Turing machine, as stated in (Eberbach, Wegner, 2003). So, agents and multi-agent systems might be considered as very powerful computational devices and may contribute with many innovative concepts to our traditional picture of the (theoretical) computer science and engineering.

An important dimension of the agent paradigm consists in considering agents not only as products of the development of computer programming techniques and as innovative tools for computer use, but also as products of development of electro-, mechanical-, and computer-engineering, as electro-mechanical (usually computer guided) devices for automation of different physical processes – as real autonomous machines which do physical (mechanical) work. From such a point of view, as we have mentioned already, there exists an important difference between real computers and the abstract Turing machine.

In (Sloman, 2002) is stated, for instance, that computers, as built and used, are the result o a convergence of the development of the machine- and electro-engineering, and the progress in understanding computations as processes of performing actions on symbols as the Turing machine do that. In our terminology, real computers as well as the real agents – (artificially) intelligent systems, esp. the cognitive robots – are entities which cannot be divided into their hardware and software parts without missing something fundamental (might be something which emerges) form the functioning of both of that their parts. According (Sloman, 2002), this difference makes computers useful, but Turing machines irrelevant for research in Artificial Intelligence, for instance. These two dimensions of agents – interaction with dynamically changing environment and

embodiment – converge into a new understanding of machines as embodied, autonomously sensing, acting and deliberating agents – into the form of *robots*. Looking to the Figure 2 we easily realize the substantial difference between the abstract universal computing devices equivalent with the Turing machine with respect of their computational power in over-simplified environment of symbols written on a tape, and the real robots equipped with computers programmed in order to control the behavior of these robots in the real dynamically changing physical environments.

The mentioned above difference, the properties like the autonomy and continuity of machines behavior, the relevance of embodiment, and other physical constrains and limitations (esp. the problematic concept of infinity with respect their behavior), the importance of communication between individually independent, autonomous computational units in order to achieve common goals (intentionally or as an emergent effect of their co-existence in a shared environment), etc. seems to be crucial for embodied systems like robots; cf. e.g. (Parker, 2003). Many computational processes in robots processors run continuously and autonomously in different types of environments. Good examples are computing processes running in autonomous mobile robots. When – for instance – a collision avoidance module is programmed, its role is to process the input sensory data continuously during the robot mission into the data manipulating with robots actuators in order to avoid obstacles in robots environments. Of course, all the program equipment of the robot may be decomposed into the set of interrelated programs of traditional type. However, this type of reduction does not contribute to the solution of the problem of collision avoidance at all! Instead of particular programs considered as translation of mathematical functions into some more procedural languages we must think in terms of autonomy and continuity of functioning of systems based on their ability to sense the environment and act in it.

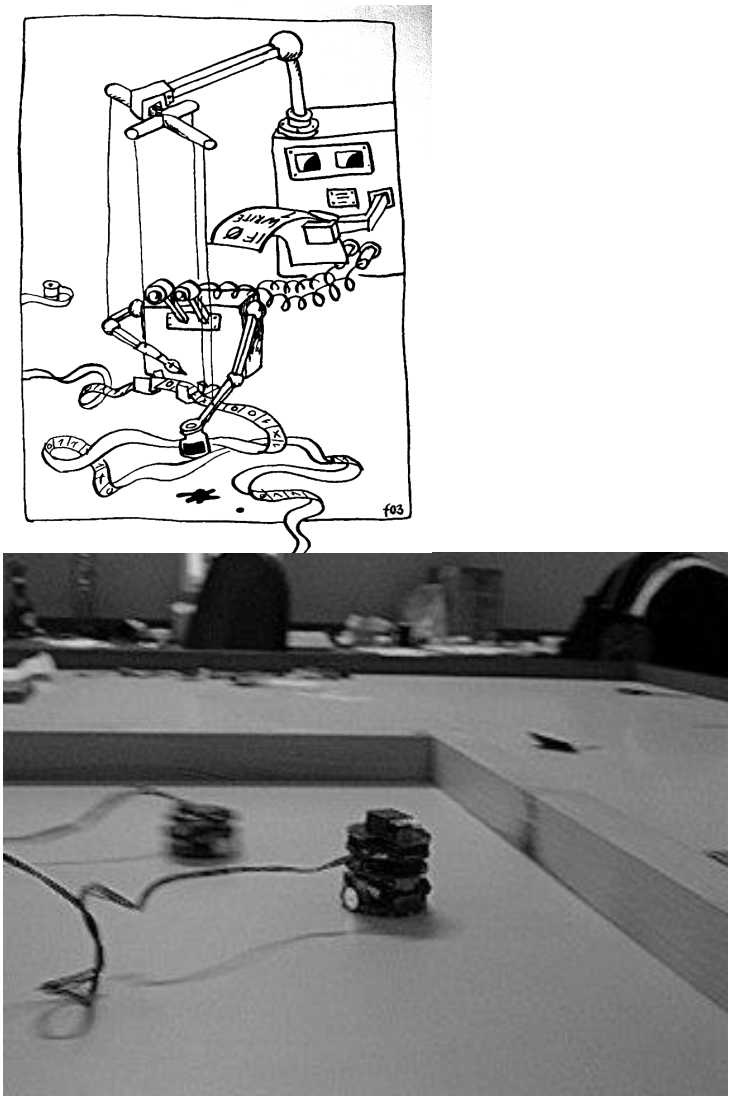

Fig. 2: Compare a particular "abstract" computing device (equivalent with the Turing machine) in its "abstract" environment from (Markoš, Kelemen, 2004), and two real embodied robots in a real dynamic environment (in Robotic Lab, Institute of Computer Science, Silesian University at Opava).

While the principal subject of the study of computation from the position of the traditional paradigm is to consider them as defined in the form of finite sequence of basic transformations executed (sequentially or in parallel) on appropriate data structures, the position of the agent paradigm might be characterize by viewing computation as the result of communication of collections of individually autonomous agents, and vice versa – viewing communication of collections of individually autonomous agents as *computation*.

In order to apply this new paradigm in modeling complicated systems (e.g. in economics, sociology, biology, robotics, etc.), the following methodological experience seems to be important: Instead of the necessity to aggregate specific particular data on individual objects as the base for (mathematical) modeling, the agent paradigm provides a tool for model each individual behavior separately and then study the emerging behavior of the society of these individuals. This methodology is present in many nowadays experiments with biological, ecological, economic, machine (robotic) or conceptual societies of agents.

5. Hyper-Computation – A Grammar-Theoretic Model

From computational point of view an appropriate sub-problem of the above described problem of embodiment is that one which consists in rigorous specification of the computational character of results of interaction of the rule governed algorithmic symbol-manipulating processes which run inside of the agents which interact massively with their computationally complicated behaving dynamic environments. Usually we are interested in as precise as possible knowledge of the behavior of an agent or a multi-agent system in its environment despite of the fact that we have no complete knowledge of the behavior of the environment. The solution of this problem is twofold: We may study the *possibility* of performing such and such behavior under such and such conditions put to the behavior of the environment. To solve this type of problem is in certain extent the traditional role of theoretical computer science. Another possibility is to concentrate to *feasibility* (of course, it will be necessary to define rigorously what we mean by feasibility in our considerations).

For instance, an *eco-grammar* (or an *EG*) *system* (Csuhaj-Varjú et al., 1997) consists of fixed number (say n) of agents with internal states described by strings of symbols w_1, w_2, ... w_n, and evolving according set of rules P_1, P_2, ..., P_n applied in a parallel way as it is usual in L systems (Rozenberg, Salomaa, 1980). The rules of agents depend, in general, on the state of the environment. The agents act in commonly shared environment (the states of the environment is described by strings of symbols w_E) by sets of sequential rewriting rules R_1, R_2, ..., R_n. The environment itself evolves according a set P_E of rewriting rules applied in parallel as in L systems. The model is schematically depicted in the figure Fig. 3.

The evolution rules of the environment are independent on agents' states and of the state of the environment itself. The agents' actions have priority over the evolution rules of the environment. In a given time unit, exactly those symbols of the environment that are not affected by the action of any agent are rewritten.

In the EG systems we assume the existence of the so called *universal clock* that marks time units, the same for all agents and for the environment, and according to which the evolution of the agents and of the environment is considered.

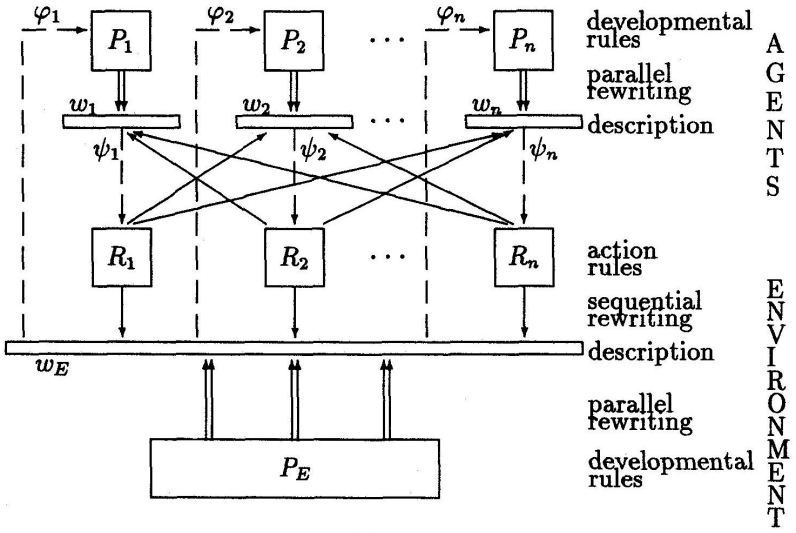

Fig. 3: A schematic view of a traditional EG system.

In (Csuhaj-varjú, Kelemenová, 1998) a special variant of EG systems have been proposed in which the agents are grouped into subsets of the set of all agents – into the so-called *teams* – and the generative power of such type of EG systems have been studied. In (Wätjen, 2003) the fixed teams are replaced by dynamically changing teams of agents. As the mechanism of reconfiguration, a function, say *f*, is defined on the set of integers with values in the same set in order to define teams. For the *i*-th step of the work of the given EG system, the function *f* relates a number *f(i)*. The subset of the set of all agents of thus EG system of the cardinality *f(i)* is the selected for executing the next derivation step of the EG system working with Wätjen-type teams. Wätjen proved, roughly speaking, that there exists an EG systems such that if *f* is in the Turing sense non-recursive function, then this EG system generates a non-recursive language. In other words, we can imagine EG systems in which subsequently changing groups of active agents interact with the dynamic environment such that the result of interaction result in the non-recursive behavior of this EG system. So, we have a grammatical model of non-recursive behavior based, similarly as in the case of the interacting Turing machines, on the interactions. This proves the hyper-computational power of interactions. The requirement of strict isolation of teams from each other, as well as the style of changes of teams during the run of the derivation process we can – at least metaphorically – interpret as the requirement of some kind of embodiment (or "embodiment"). So, a system of some simple (finite) derivative units – agents – created in some complicated (non-computable) way which interact with a specific shared dynamic environment provide the hyper-computational power of the behavior of the whole system set up from the agents and their environment.

The range of question (with certain relevance to Artificial Life or to the computational studies of evolution, for instance) we can ask about an EG system contains e.g. the following items:

- Is the evolution of a system bounded in time or not? In other words: Enters the system a deadlock? When, under what conditions?
- In the case of the infinite evolution, are non-cyclic evolution chains possible?
- What is the effect of „small changes" either in the initial configuration or in the evolution/action rules of agents and of the evolution rules of the environment?
- What is the effect of introducing further life-like features (the Wätjen-type teams are naturally interpretable in the frame of Artificial Life) into the model to the answers to the previous questions?
- What is the influence of the number of agents on the system properties?

The role of the body – at least in the case when we discuss it from computational point of view – consists first of all in generating the behavior of the embodied system with respect the situation appearing in its environment. In order to study the conditions and the power of the generation of behaviors of this type formally we require at least the following:

- to have an opportunity to study the system with respect of its constituent parts and their interactions, and
- to have an opportunity to study the interaction of the whole system or its parts with their environments.

In such a situation, we may ask the following questions, for instance:

- What is the computational power of the EG system working on the environment generated by a super-recursive device?
- Are EG systems "regulate the super-computable behavior of their environment into the form of a computable one? Under what kind of circumstances?

Let we suppose now that we are able construct on the base of our knowledge in theoretical computer science and with respect the Turing hypothesis only devices with behaviors computable in the sense of the traditional Turing-computability. Suppose that a good theoretical framework for describing this type of devices is the framework of some variant of EG systems. Suppose that the behavior of the environment of EG systems might be very unconventional, exotic ones. Suppose it might be un-computable in the sense of the traditional Turing-computability; for other alternatives see e.g. (Eberbach, Wegner, 2003). Why not? Technically it means that instead of an L system supposed as the generative base for autonomous changes of the environment in a given EG system we suppose that the environment may change in some non-computable in the sense of Turing

computability manners. What can we say concerning the behavior of such type of EG systems? More technically: What type of the behavior of an EG system we may expect when this system works in a non-computably changing environment?

Concerning the study of feasibility, we may be interested not only in the exact knowledge of the behavior of an EG system, but also in the (importance of) difference between the behavior of an EG system and the required behavior of it.

References:

Ačová, M., Kelemen, J., Kubík, A.: Embodied hypotheses – preliminary notes and case studies. In: *Proceedings of the RoMoCo 04 – The 4th International Workshop on Robot Motion and Control* (K. Kozlowski, Ed.) Poznan University of Technology, Poznan, 2004, pp. 35-40

Brooks, R. A.: *Cambrian Intelligence*. The MIT Press, Cambridge, Mass., 1999

Brooks, R. A.: The relationship between matter and life. *Nature* **406** (2001) pp. 409-4

Burgin, M., Klinger, A.: Preface – three aspects of super-recursive algorithms and hyper-computation or finding black swans. *Theoretical Computer Science* **317** (2004) 1-11

Copeland, B. J.: Hypercomputation – some philosophical issues. *Theoretical Computer Science* **317** (2004) 251-267

Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Paun, Gh.: Eco-grammar systems – a grammatical framework for lifelike interactions. *Artificial Life* **3** (1997) 1-28

Csuhaj-Varjú, E., Kelemenová, A.: Team behaviour in eco-grammar systems. *Theoretical Computer Science* **209** (1998) 213-224

Eberbach, E., Wegner, P.: Beyond Turing machines. *Bulletin of the EATCS* **81** (2003) 279-304

Ernst, G., Newell, A.: *GPS – A Case Study in Generality and Problem Solving*. Academic Press, New York, 1969

Ferber, J.: *Multi-Agent Systems*. Addison-Wesley, Harlow, 1999

Freud, S.: Project for a scientific psychology. In: *The Origins of Psycho-Analysis* (M. Bonaparte, A. Freund, and E. Kris, eds.). Basic Books, New York, 1954

Horáková, J., Kelemen, J.: Čapek, Turing, von Neumann, and the 20th century evolution of the concept of machine. In: *Proceedings of the International Conference in Memoriam John von Neumann*. Budapest Polytechnic, Budapest, 2003, pp. 121-135

Humphrey, N.: *How to Solve the Mind-Body Problem*. Imprint Academic, Thorverton, 2000

Kelemen, J.: The agent paradigm – foreword. *Computing and Informatics* **22** (2003) 513-519

Markoš, A., Kelemen, J.: *Berušky, andělé a stroje* (Ladybugs, Angels, and Machines; in Czech). Dokořán, Prague, 2004

Minsky, M.: *The Society of Mind*. Simon and Schuster, New York, 1986

Newell, A.: *Unified Theories of Cognition*. Harvard University Press, Cambridge, Mass., 1990

Newell, A., Simon, H. A.: *Human Problem Solving*. Prentice Hall, Englewood Cliffs, N. J., 1972

Parker, L.: Current research in multirobot systems. *Artificial Life and Robotics* **7** (2003) 1-5

Panhuysen, G.: The relationship between somatic and psychic processes – lessons from Freud's *Project*. In: *Neuroscience of the Mind on the Centennial of Freud's Project for a Scientific Psychology* (R. M. Bilder, and F. F. LeFever, eds.). Annals of the New York Academy of Sciences, vol. 843. NYAS, New York, 1998, pp. 20-42

Rozenberg, G, Salomaa, A.: *The Mathematical Theory of L-systems*. Academic Press, New York, 1980

Scheutz, M.: Computationalism – the next generation. In: *Computationalism – New Directions* (M. Scheutz, Ed.). The MIT Press, Cambridge, Mass., 2002, pp. 1-21

Sloman, A.: The irrelevance of Turing machines to artificial intelligence. In: *Computationalism – New Directions* (M. Scheutz, Ed.). The MIT Press, Cambridge, Mass., 2002, pp. 87-127

Stefik, M.: *Introduction to Knowledge Systems*. Morgan Kaufmann Publ., San Francisco, Cal., 1995

Tecuci, G.: *Building Intelligent Agents*. Academic Press, San Diego, Cal., 1998

Wätjen, D.: Function-dependent teams in eco-grammar systems. *Theoretical Computer Science* **306** (2003) 39-53

Wegner, P.: Why interaction is more powerful than algorithms. *Communications of the ACM* **40** (1997) No. 5, 81-91

Wegner, P., Goldin, D.: Computing beyond Turing machines. *Communications of the ACM* **46** (2003) No. 4, 100-102

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 240 - 254, 2004.

Monocultures and Homogeneous Environment in Eco-Grammar Systems*

Alica Kelemenová, Michal Tupý
Institute of Computer Science, Silesian University
Opava, Czech Republic
kelemenova@fpf.slu.cz, Michal.Tupy@seznam.cz

1 Introduction

The model of an eco-grammar system was introduced in [6] and presented in detail in [7]. It realizes an attempt to create formal specification for investigation of the interplay between the environment and agents in systems like ecosystems. Eco-grammar systems can be used to model some aspects of the behaviour of any cooperating communities of agents acting in a common dynamic environment. The model is based on the approaches and methods of formal language theory using generative framework of grammar systems [2].

Basic information on the topic can be found in overview papers [5], [11], [12], [13] and [14]. Annotated bibliography *www.sztaki.hu/mms/ecobib.html* provides a good source of information on eco-grammar systems.

The model consists of two interconnected parts. The environment, described by a string developing by L system mode, in totally parallel manner using interactionless rules, and collection of agents (components), each one described by its own string, developing by L system mode using its own set of rules. The agent locally changes the environment using its action rules. Actual state of the environment can influence development of agents and the state of each agent can influence the development of the environment by the choice of the action rules.

The original model is described in a quite general way in order to demonstrate wide possibility of the model to investigate various aspects of the behaviour of ecosystems and to present formal background for such an investigation.

To get insight to the behaviour of eco-grammar systems different special cases, characterized by various restrictions, were studied. We mention simple eco-grammar systems, for example, where agents influence the development of the environment just by their action rules (but not by the states of agents). (See [2], [4], [8], [9], [17].)

In the present paper we study eco-grammar systems, where the behaviour of the environment is really influenced by the state of agents (not only by the existence of agents and their action rules). We discuss in detail the generative power of

*Research supported by the Grant Agency of the Czech Republic grant No. 201/04/0528

monocultures and the generative power of eco-grammar systems with homogeneous environments. The results are related to the number of components of the systems.

In the Section 2 we give formal definition of the eco-grammar system and fix notations used in the paper.

Section 3 contains various examples of eco-grammar systems and some preliminary results concerning the properties of eco-grammar systems generating some typical languages of eco-grammar systems.

In Section 4 we start to study systems with identical components, called monocultures. The restriction to monocultures *with fixed number of components* restricts also the generative power of eco-grammar systems. An important result is that any language of eco-grammar system can be transformed to a language of monoculture simply by adding one word to the original language. Corresponding eco-grammar system for the monoculture have the same number of components as the original one. Still open problem remains the equality of the languages classes of monocultures and the (general) eco-grammar languages when no restriction to the number of components is considered. Number of components is important for the generative power of monocultures. Language classes with different number of components are incomparable.

In Section 5 we consider systems with homogeneous environment (systems with environment described by strings over the unary alphabet). In this case number of components in eco-grammar systems introduces full hierarchy on the corresponding language classes. Eco-grammar systems with less components are more powerful. Languages of unary eco-grammar systems can be generated by eco-grammar systems with single component.

In Section 6 the unary monocultures are treated, i.e. eco-grammar systems with identical components and unary alphabet of the environment. Unary monocultures are as powerful (generative power) as unary eco-grammar systems. This solves an open problem from Section 4 for unary monocultures. The number of components in unary monocultures introduces partial hierarchy on the corresponding language classes, depending on the fact, whether the number of components of the one of the system is a multiple of the number of components of the other system.

In all previous results context rules are used for action of the components in the environment. Consequences of the restriction to context-free action rules are discussed in Section 7. Eco-grammar systems with context-free action rules are less powerful as those with context action rules. The restriction to the context-free case leads to different results for the problems we are dealing with. Language classes of eco-grammar systems with different number of components are incomparable. Monocultures with context-free actions are less powerful as general context-free eco-grammar systems. Last result holds even if no restriction to the number of components are treated. Each context-free eco-language can be extended by one word to a language of context-free monoculture. Incomparability of language classes with different size of components is presented, too.

2 Eco-Grammar Systems: Original Model and its Special Versions

The models of eco-grammar systems are introduced in [6], [7]. An eco-grammar system Σ consists of two mutually influencing parts, an environment characterized by an $0L$ scheme $E = (V_E, P_E)$ and a collection of n agents $A_i = (V_A, P_i, \varphi_i, R_i, \psi_i)$, $1 \leq i \leq n$ characterized by inner developmental rules P_i used to develop i-th agent, action rules R_i influencing the environment locally in the chosen place. Selection functions φ_i, ψ_i determine actually active rules.

Definition 2.1 *An eco-grammar system of the degree n (EG(n) for short) is a structure* $\Sigma = (E, A_1, A_2, \ldots, A_n)$, *where*

- $E = (V_E, P_E)$ *is an 0L schema called an environment of the EG(n)*
- $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i, w_i)$, *for* $1 \leq i \leq n$ *is i-th agent of EG(n), where*
  - V_i *is a finite alphabet,*
  - P_i *is a finite complete set of context-free rules over* V_i,
  - R_i *is a finite set of rules over* V_E,
  - φ_i *is a function* $V_E^* \to P(P_i)$,
  - ψ_i *is a function* $V_i^+ \to P(R_i)$,
  - w_i *is an axiom,* $w_i \in V_i^*$.

Definition 2.2 A configuration *(or state) of EG(n) system* Σ *is an (n+1)-tuple* $\langle v \rangle = (v_E, v_1, v_2, \ldots, v_n)$, *where* v_E *is a state of the environment and* v_i *for* $1 \leq i \leq n$ *is a state of the i-th agent. The starting configuration is an* $(n+1)$*-tuple* $\langle w \rangle = (w_0, w_1, w_2, \ldots, w_n)$, *where* w_0 *is a starting state (an axiom) of the environment and* $w_1, w_2, \ldots, w_n$ *are axioms of the agents* $A_1, A_2, \ldots, A_n$, *respectively.*

Definition 2.3 A derivation step *in the EG(n) system* $\Sigma = (E, A_1, A_2, \ldots, A_n)$ *is a binary relation* $\Longrightarrow_\Sigma$ *over* $V_E^* \times V_1^* \times V_2^* \times \ldots \times V_n^*$ *such that*
$(v_E, v_1, v_2, \ldots, v_n) \Longrightarrow_\Sigma (v'_E, v'_1, v'_2, \ldots, v'_n)$ *if and only if*

- $v_E = \alpha_0 \beta_{i_1} \alpha_1 \beta_{i_2} \ldots \alpha_{n-1} \beta_{i_n} \alpha_n$,
  $v'_E = \alpha'_0 \beta'_{i_1} \alpha'_1 \beta'_{i_2} \ldots \alpha'_{n-1} \beta'_{i_n} \alpha'_n$,
  where
  $\alpha_k \Longrightarrow_E \alpha'_k, 0 \leq k \leq n$,
  $\beta_{i_k} \to \beta'_{i_k} \in \psi_{i_k}(v_{i_k}), 1 \leq k \leq n, \{i_1, \ldots, i_n\} = \{1, 2, \ldots, n\}$.
- $v_i \Longrightarrow_{\varphi_i(v_E)} v'_i, 1 \leq i \leq n$.

We assume that all agents are active in any derivation step (expressed by $\{i_1, \ldots, i_n\} = \{1, 2, \ldots, n\}$).

In the case when it is clear which $_\Sigma$ is considered the symbol $_\Sigma$ can be omitted. A transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^+$ and a transitive and reflexive closure of the relation $\Longrightarrow$ is denote by $\Longrightarrow^*$.

A derivation is in *a canonic form*, if moreover

$$\begin{aligned} v_E &= \beta_1\beta_2\ldots\beta_n\alpha_n, \\ v'_E &= \beta'_1\beta'_2\ldots\beta'_n\alpha'_n. \end{aligned}$$

In the case when the choice of an action rule is independent on the actual state of agent A, i.e. $\psi_A(v) = R_A$ for all $v \in V_A^+$ the environment is independent on actual states of the agents and agents influence the development of the environment by the set of its action rules only. The selection function for activity of agents is universal (any action rule can be active). In this case we specify agents simply by $A = \{R_A\}$. Such an agent is called *a simple agent*. An eco-grammar system with simple agents only is called *simple eco-grammar system*.

A configuration of the simple eco-grammar system consists only of the state of the environment. A derivation step $\Longrightarrow_\Sigma$ over V_E^* of the simple eco-grammar system $\Sigma = (E, A_1, A_2, \ldots, A_n)$ has the form $v \Longrightarrow_\Sigma v'$ if and only if
$v = \alpha_0\beta_{i_1}\alpha_1\beta_{i_2}\ldots\alpha_{n-1}\beta_{i_n}\alpha_n$, and $v' = \alpha'_0\beta'_{i_1}\alpha'_1\beta'_{i_2}\ldots\alpha'_{n-1}\beta'_{i_n}\alpha'_n$, for
$\alpha_k \Longrightarrow_E \alpha'_k, 0 \le k \le n$, and $\beta_{i_k} \to \beta'_{i_k} \in R_{i_k}, 1 \le k \le n, \{i_1, \ldots, i_n\} = \{1, 2, \ldots n\}$.

Definition 2.4 A language *of an EG(n) system* $\Sigma = (E, A_1, A_2, \ldots, A_n)$ *and the initial state of the environment* w_0 *is a language of all states of the environment which can be derived from the initial configuration, i.e.*
$L(\Sigma, w_0) = \{v : (w_0, w_1, w_2, \ldots w_n) \Longrightarrow^* (v, v_1, v_2, \ldots v_n)\}$.

Definition 2.5 *An EG(n) system* Σ *is* monoculture *(MEG(n) for short) if* $A_i = A$ *for all* $1 \le i \le n$. *We shall use the denotation* $\Sigma = (E, A^n)$ *in this case.*

An EG(1) system is evidently monoculture.

Definition 2.6 *An EG(n) system* Σ *is* unary *(UEG(n)) if* $|V_E| = 1$. *Unary monoculture with n agents is denoted by MHEG(n).*

Definition 2.7 *The language class of eco-grammar systems (monoculture, unary, unary monoculture) of the degree n is denoted by* $\mathcal{L}(EG(n))$ *(*$\mathcal{L}(MEG(n))$*,* $\mathcal{L}(UEG(n))$*,* $\mathcal{L}(MUEG(n))$*).*

3 Systems with Locally Restricted Exponential Behavior

The language $L_{l,m} = \{a^{l^i+m} : i \in N\}$ is frequently used in the further parts of the paper. Its typical property is that each underlining EG system for $L_{l,m}$ has exactly one rule in the environment.

Lemma 3.1 *Let* $L_{l,m} = L(\Sigma, w_0)$ *for an EG(n) system* $\Sigma = (E, A_1, \ldots, A_n)$. *Then* $P_E = \{a \to a^{l^j}\}$ *for some* $j \ge 1$.

Proof: Assume contrary, that two different rules $a \to a^r, a \to a^s$ are in P_E and $r > s$. There are infinitely many words in $L(\Sigma, w_0)$ with at least one symbol rewritten

by the rules of the environment. Therefore infinitely many pairs of words u, v in $L(\Sigma, w_0)$ have $|u| - |v| = r - s$. But $L_{l,m}$ does not have that property.
Let $P_E = \{a \to a^s\}$ for some fixed number s. We have $s \neq 0, s \neq 1$ since $L(\Sigma, w_0)$ is finite for $P_E = \{a \to \lambda\}$ and there are infinitely many pairs of words $u, v \in L(\Sigma, w_0)$ such that $u \Longrightarrow v$, with constant $|u| - |v|$ for $P_E = \{a \to a\}$.
Assume $s = l^j + d$ for some fixed j and d, $0 \leq d < l^{j+1} - l^j$. We prove that $d = 0$.
Let $(a^k, w_1, w_2, \ldots, w_n) \Longrightarrow (a^{k'}, w'_1, w'_2, \ldots, w'_n)$ for $k = l^i + m$ and $k' = l^{i'} + m$.
Let $r \geq n$ letters are rewritten to t letters by action rules and all other letters are rewritten by the rule of the environment in that derivation step. Then

$$\begin{aligned} k' &= ks - rs + t, \\ l^{i'} + m &= (l^i + m) * s - rs + t, \end{aligned}$$

Now we assign $c' = sm - rs + t - m$. All numbers are constant (s, m) or limited (r, t) so c' is limited too.

$$\begin{aligned} l^i * s + c' &= l^{i'}, \\ l^i(l^j + d) + c' &= l^{i'}, \\ l^{i+j} + l^i * d + c' &= l^{i'}. \end{aligned}$$

We can choice i such that $l^i * d + c' > 0, l^i > max(|c'|)$ so

$$l^{i+j} + l^i * d + c' > l^{i+j}$$

and so for $l^j - l^{j-1} - 1 \geq d$ we obtain

$$l^{i+j} + l^i * d + c' \leq l^{i+j+1} - l^i + c' < l^{i+j+1}.$$

No i' exists so $d = 0$ and $P_E = \{a \to a^{l^j}\}$ for some $j \geq 1$. □

Example: Various representations of the language $L_{l,m} = \{a^{l^i+m} : i \in N\}$ by EG(m) systems are possible. They differ in the rule of the environment.

$$\begin{aligned} \Sigma_j &= (E, A_1, A^{m-1}), \\ E &= (\{a\}, \{a \to a^{l^j}\}), \\ A_1 &= (V, P, R, \varphi, \psi, Y), \\ A &= (V, P, R, \varphi, \psi, X), \\ V &= \{X, Y\}, \\ P &= \{X \to X, Y \to X\}, \\ R &= \{a \to a\} \cup \{a^{l+1} \to a^{l^i+1} : 1 \leq i \leq j\}, \\ \varphi(w) &= P, \text{ for } w \in a^* \\ \psi(Y) &= \{a^{l+1} \to a^{l^i+1} : 1 \leq i \leq j\}, \\ \psi(X) &= \{a \to a\}. \end{aligned}$$

Derivations in the system are of the form
$(a^{l+m}, Y, X, \ldots, X) \Longrightarrow (a^{l^i+m}, X, X, \ldots, X) \Longrightarrow (a^{l^i l^j+m}, X, X, \ldots, X) \Longrightarrow$
$(a^{l^i l^{2j}+m}, X, X, \ldots, X) \Longrightarrow \ldots$

In the first step, the first agent replaces $l + 1$ letters by one of the words $a^{l^1+1}, \ldots, a^{l^j+1}$ and each of the remaining $m - 1$ agents leave symbol a unchanged.

After the first step one of the words $a^{l+m}, \ldots, a^{lj+m}$ is generated. In next steps agents leave m symbols unchanged and a^{l^i} symbols are rewritten by the rule of the environment. So $L(\Sigma, a^{l+m}) = L_{l,m}$.

Lemma 3.2 *Let $L = \{w_1, \ldots, w_n\}, n \geq 2$ and $|w_i| \leq k-1$ for all $1 \leq i \leq n$. Then L is not an EG(k) language.*

Proof: Each agent of an EG(k) system rewrites at least one symbol of the environment. Any language with at least two words produced by an EG(k) system has to have the word of the system of the length at least k, which is chosen to be the axiom. Otherwise no word can be generated from the axiom. □

4 Monocultures

Monocultures are eco-grammar systems with identical agents including their axioms. Obviously, each system with one agent is a monoculture. Monocultures with at least two agents are less powerful than eco-grammar systems with the same number of agents.

Theorem 4.1 *$\mathcal{L}(MEG(n)) \subset \mathcal{L}(EG(n))$ for $n > 1$.*

Proof: We have $\mathcal{L}(MEG(n)) \subseteq \mathcal{L}(EG(n))$ by the definition.
For $n > 1$ $L = \{a^{n-1}, a^n\} \in \mathcal{L}(EG(n)) - \mathcal{L}(MEG(n))$ and simple EG(n) system $\Sigma = ((\{a\}, \{a \to a\}), \{a \to \lambda\}, \{a \to a\}^{n-1})$ with axiom a^n generates $L(\Sigma, a^n) = L$.
We prove $L \notin \mathcal{L}(MEG(n))$ by contradiction. Suppose that there is an MEG(n) system $\Sigma' = (E', A'^n), A' = (V', P', R', \varphi', \psi', w_0')$ producing L.
The axiom of the environment is a^n since each agent rewrites in one step at least one letter. To obtain a^{n-1} one of the agents acts with the rule $a \to \lambda$. But, alternatively, the same rule $a \to \lambda$ can be used by all agents, too, and the system Σ' produces λ contradictory with the form of L. □

The situation may be different in a case, where no limitation to the number of agents is considered.

Open problem: (In)equality of classes $\mathcal{L}(EG)$ and $\mathcal{L}(MEG)$.

It is surprising that there is not too big difference between generative power of the eco-grammar systems and monocultures. Indeed, an arbitrary language of eco-grammar system can be transformed to language of monoculture by adding one special word to it.

Theorem 4.2 *Let $L \in \mathcal{L}(EG(n)) - \mathcal{L}(MEG(n))$. Then there is a word u such that $L \cup \{u\} \in \mathcal{L}(MEG(n))$.*

Proof: Let $L \in \mathcal{L}(EG(n)) - \mathcal{L}(MEG(n))$ and $L = L(\Sigma, w)$ for axiom w and for $\Sigma = (E, A_1, A_2, \ldots, A_n)$, $E = (V_E, P_E)$, $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i, w_{i,0})$.

Assume $V_i \cap V_j = \emptyset$ for $i \neq j$ and $|V_E| > 1$. Let us choose any $u = u_1 u_2 \dots u_n$ such that $u_i \in V_E^+$ and $|u|_{u_i} = 1$ for $i = 1, \dots, n$. (In the case $|V_E| = 1$ we extend the alphabet to the binary one, in order to get an axiom u with the above property.)

We construct an MEG system $\Sigma' = (E, A^n)$, $A = (V, P, R, \varphi, \psi, a)$ such that $L' = L \cup \{u\} = L(\Sigma', u)$:

$$\begin{array}{rcl}
V & = & V_1 \cup V_2 \cup \dots \cup V_n \cup \{a\} \cup \{a_i : 1 \leq i \leq n\} \text{ for} \\
 & & (V_1 \cup V_2 \cup \dots \cup V_n) \cap (\{a\} \cup \{a_i : 1 \leq i \leq n\}) = \emptyset, \\
P & = & P_1 \cup P_2 \cup \dots \cup P_n \cup \{a \to a_i : 1 \leq i \leq n\} \cup \\
 & & \cup \{a_i \to w_{i,0} : 1 \leq i \leq n\}, \\
R & = & R_1 \cup R_2 \cup \dots \cup R_n \cup \{u_i \to u_i : 1 \leq i \leq n\} \cup \\
 & & \cup \{u_1 \to w\} \cup \{u_i \to \lambda : 2 \leq i \leq n\}, \\
\varphi : & & \varphi(u) = \{a \to a_i, a_i \to w_{i,0} : 1 \leq i \leq n\}, \\
 & & \varphi(v) = \varphi_1(v) \cup \varphi_2(v) \cup \dots \cup \varphi_n(v) \text{ for } v \in L, \\
\psi : & & \psi(a) = \{u_i \to u_i : 1 \leq i \leq n\}, \\
 & & \psi(a_1) = \{u_1 \to w\}, \\
 & & \psi(a_i) = \{u_i \to \lambda : 2 \leq i \leq n\}, \\
 & & \psi(x) = \psi_i(x) \text{ for all } x \in V_i^*.
\end{array}$$

The derivation in Σ' proceeds as follows:

$$(u, a, a, \dots, a) \Longrightarrow_{\Sigma'} (u, a_1, a_2, \dots, a_n) \Longrightarrow_{\Sigma'} (w, w_{1,0}, w_{2,0}, \dots, w_{n,0})$$

Σ' is in starting configuration of system Σ generating language $L(\Sigma, w)$ and next steps follow derivation in Σ, therefore $L(\Sigma', u) = L(\Sigma, w) \cup \{u\}$. □

Theorem 4.3 *$\mathcal{L}(MEG(n))$ and $\mathcal{L}(MEG(m))$ are incomparable for $n \neq m$.*

Proof: Let $n > m$.

a) $L_{n,m} = \{a^{n^i+m} : i \in N\} \in \mathcal{L}(MEG(m)) - \mathcal{L}(MEG(n))$.

For MSEG(m) system $\Sigma = ((\{a\}, \{a \to a^n\}), \{a \to a\}^m)$ we have $L(\Sigma, a^{1+m}) = L_{n,m}$.

Suppose there is an MEG system Σ with $n, n > m$ agents for $L_{n,m}$. In the first derivation step n agents can use the same rule $x \to x'$ and all letters in the remaining part of the environment are rewritten to the power of n letters by Lemma 3.1. Therefore the axiom $w_0 = x^n y$ is rewritten to $w' = (x'^n y'^n)$, $|x'|*n + |y'|*n = kn$, i.e. $n^i + m = kn$ for some k and this is not valid for $n > m > 0$.

b) $L_n = \{(a^i b^i)^n : i \geq 1\} \in \mathcal{L}(MEG(n)) - \mathcal{L}(MEG(m))$. The MEG(n) system $\Sigma = (E, A^n)$ for L_n has $E = (\{a, b\}, \{a \to a, b \to b\})$, $A = \{ab \to aabb\}$ and the axiom $(ab)^n$.

To prove $L_n \notin \mathcal{L}(MEG(m))$ for $n > m$ assume contrary that there is an MEG(m) system $\Sigma' = (E', A'^m)$ and $L(\Sigma', w) = L_n$ for some axiom w. We prove that $P_{E'} = \{a \to a^i, b \to b^i\}$ for some $i \in N$.

The environment does not contain a rule with both symbols a, b on the right side. Otherwise some derived word contains more than n occurrences of subword ab or ba.

The environment contains neither rules $a \to b^i$ nor $b \to a^j$. Otherwise some derived word starts with b or ends with a.

The environment contains neither a pair of rules $a \to a^i, a \to a^j$ nor pair of rules $b \to b^i, b \to b^j$ for $i \neq j$. Otherwise both $a^i w', a^j w' \in L(\Sigma', w))$ or both $w'b^i, w'b^j \in L(\Sigma', w)$ and L_n does not have such a property.

The environment does not contain rules $a \to a^i, b \to b^j$ for $i \neq j$, otherwise the language $L(\Sigma', w)$ contains words w with $|w|_a \neq |w|_b$.This gives $E' = (\{a, b\}, \{a \to a^i, b \to b^i\})$ for some $i \geq 1$.

Let r be maximum number of symbols rewritten by agents and t be maximum number of symbols generated by agents in one derivation step. Let us consider a derivation step $(w, w_1, \ldots, w_m) \Longrightarrow (w', w'_1, \ldots, w'_m)$ for $w = (a^h b^h)^n, h > r, h > t$. The value of h guarantees that action rules $u \to v$ used to rewrite $(a^h b^h)^n$ have $u, v \in a^* b^* \cup b^* a^*$. Each agent can change some symbols in one neighboring occurrences of subwords a^h, b^h. At least one occurrence of a^h or b^h is rewritten by the rules of the environment. Assume that rules of agents are applied in the prefix $(a^h b^h)^m a^h$ of w in the discussed derivation step $((a^h b^h)^n, w_1, \ldots, w_m) \Longrightarrow (w', w'_1, \ldots, w'_m)$. Then the last occurrence of the string b^h in w is rewritten by the rules of the environment and we have $w' = (a^{h'} b^{h'})^n = v' a b^{hi}$. For $i = 1$, $h' = h$, $w' = w$ and the language $L(\Sigma', w))$ is finite. For $i > 1$, $h' = 2h$ and $(a^{h+1} b^{h+1})^n \notin L(\Sigma', w)$. Therefore $L \notin \mathcal{L}(MEG(m))$.

c) $a^* a^n$ and $\{a^{m-1}\}$ are examples of languages both in $\mathcal{L}(MEG(m))$ and also in $\mathcal{L}(MEG(n))$ for $n > m > 0$. □

5 Unary Eco-Grammar Systems

In this section systems with homogeneous environment represented by unary alphabet are studied. We compare generative power of unary eco-grammar systems with different number of components. These language classes are ordered by inclusion relation. The power of systems increases by decreasing the number of agents.

Theorem 5.1 *$\mathcal{L}(UEG(n+1)) \subset \mathcal{L}(UEG(n))$ for $n \geq 1$.*

Proof: To prove $\mathcal{L}(UEG(n+1)) \subseteq \mathcal{L}(UEG(n))$ assume a UEG(n+1) system $\Sigma = (E, A_1, A_2, \ldots A_n, A_{n+1})$, with $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i, w_{i,0})$ for $1 \leq i \leq n+1$. We construct an equivalent UEG(n) system Σ', where the last component of Σ' simulates both the behaviour of n-th and $n+1$-st component of Σ.

$\Sigma' = (E, A_1, \ldots, A_{n-1}, B)$, where $B = (V_B, P_B, R_B, \varphi_B, \psi_B, w_{B,0})$ and

$$\begin{aligned}
V_B &= V_n \cup V'_{n+1}, \\
P_B &= P_n \cup P'_{n+1}, \\
R_B &= \{\alpha_n \alpha'_{n+1} \to \beta_n \beta'_{n+1} : \alpha_n \to \beta_n \in R_n, \alpha_{n+1} \to \beta_{n+1} \in R_{n+1}\}, \\
\varphi_B &= \varphi_n \cup \varphi'_{n+1}, \\
\psi_B(w_n w'_{n+1}) &= \{a^k \to a^l : k = k_n + k_{n+1}, l = l_n + l_{n+1}, a^{k_n} \to a^{l_n} \in \psi_n(w_n), \\
&\quad a^{k_{n+1}} \to a^{l_{n+1}} \in \psi_{n+1}(w_{n+1})\}, \\
w_{B0} &= w_{n,0} w'_{n+1,0}.
\end{aligned}$$

By $V'_{n+1}, P'_{n+1}, \alpha'_{n+1}, \beta'_{n+1}, w'_{n+1,0}$ we mean primed copy of the original objects.

$L(\Sigma, w_0) = L(\Sigma', w_0)$ follows from the property:
$(w_i, w_{1,i}, \ldots, w_{n,i}, w_{n+1,i}) \Longrightarrow_\Sigma (w_{i+1}, w_{1,i+1}, \ldots, w_{n,i+1}, w_{n+1,i+1})$
if and only if
$(w_i, w_{1,i}, \ldots, w_{n,i} w'_{n+1,i}) \Longrightarrow_{\Sigma'} (w_{i+1}, w_{1,i+1}, \ldots, w_{n,i+1} w'_{n+1,i+1})$.

Σ and Σ' are unary, therefore place where agents act do not influence the resulting word. Assume that agents rewrite the prefix of the actual environmental word and that developmental rules of environment are used to rewrite its suffix. This gives identical environmental words derived by systems Σ and Σ'.

$\{a^n, a^{n-1}\} \in \mathcal{L}(UEG(n)) - \mathcal{L}(UEG(n+1))$ implies proper inclusion. □

Important consequence of the previous theorem is that unary eco-grammar systems with one component can generate all UEG languages.

Theorem 5.2 $\mathcal{L}(UEG(1)) = \mathcal{L}(UEG)$

Proof: $\mathcal{L}(UEG(n)) \subset \mathcal{L}(UEG(1))$ for each $n \geq 2$ directly follows from the Theorem 5.1 and

$\mathcal{L}(UEG) = \bigcup_{i=1}^{\infty} \mathcal{L}(UEG(n)) \subseteq \mathcal{L}(UEG(1)) \subseteq \mathcal{L}(UEG)$.

So all $\subseteq$ has to be equality, which gives the Theorem. □

6 Unary Monocultures

According the previous Sections eco-grammar systems with homogeneous environments posses hierarchy on the language classes defined by systems with different number of components, while monocultures with different number of components are incomparable with respect to the generative power. In the present Section we show that homogeneous monocultures introduce partial ordering on the language classes of eco-grammar systems with different number of components.

Theorem 6.1 *Let m, n be natural numbers $n > m > 0$.*

$\mathcal{L}(UMEG(n)) \subset \mathcal{L}(UMEG(m))$ for m dividing n,

$\mathcal{L}(UMEG(n))$ and $\mathcal{L}(UMEG(m))$ are incomparable, otherwise.

Proof: Let $n = cm$ for $c > 1$. We prove $\mathcal{L}(UMEG(cm)) \subseteq \mathcal{L}(UMEG(m))$. Let UMEG(cm) system $\Sigma = (E, A^{cm}), A = (V, P, R, \varphi, \psi, w_0)$ produces $L = L(\Sigma, w)$. We construct an UMEG(m) system $\Sigma' = (E, B^m)$, $B = (V', P', R', \varphi', \psi', w'_o)$ such that $L(\Sigma', w) = L(\Sigma, w) = L$:

$$\begin{aligned}
V' &= V \cup \{@\}, \\
P' &= P \cup \{@ \to @\}, \\
R' &= \{\alpha_1 \ldots \alpha_c \to \alpha'_1 \ldots \alpha'_c : \alpha_i \to \alpha'_i \in R, 1 \leq i \leq c\}, \\
\varphi'(w_E) &= \varphi(w_E) \cup \{@ \to @\}, \\
\psi'(w_1 @ w_2 @ \ldots @ w_c) &= \{\alpha_1 \alpha_2 \ldots \alpha_c \to \alpha'_1 \alpha'_2 \ldots \alpha'_c : \alpha_i \to \alpha'_i \in \psi(w_i), 1 \leq i \leq c\}, \\
w'_0 &= (w_0 @)^{c-1} w_0.
\end{aligned}$$

Agent B simulates the behaviour of c agents A with respect to the environment, so $L(\Sigma, w) = L(\Sigma', w)$.

The above inclusion is proper. To prove it we verify $L_{n,m} = \{a^{n^i+m} : i \in N\} \in \mathcal{L}(UMEG(m)) - \mathcal{L}(UMEG(n))$.

There is a UMEG(m) system $\Sigma = ((\{a\}, \{a \to a^n\}), \{a \to a\}^m)$ such that $L(\Sigma, a^{n+m}) = L_{n,m}$.

Suppose there is an UMEG(n) system Σ for $L_{n,m}$. In the first derivation step n agents can use the same rule and all letters in the remaining part of the environment are rewritten to the power of n as in Lemma 3.1. Therefore $w_0 = x^n y$ is rewritten to $w' = (x'^n y'^n)$, $|x'| * n + |y'| * n = kn$, i.e. $n^i + m = kn$ and this is not valid for $n > m > 0$.

$$L_{m,n} = \{a^{m^i+n} : i \in N\} \in \mathcal{L}(UMEG(n)) - \mathcal{L}(UMEG(m)) \text{ for } m \text{ not dividing } n.$$

To prove $L_{m,n}$ is not in $\mathcal{L}(UMEG(m))$ analogously to the discussion above we have $m^i + n = km$. This can be satisfied just for m dividing n.

Languages a^*a^n and $\{a^{m+1}\}$ are both in $\mathcal{L}(UMEG(m))$ and $\mathcal{L}(UMEG(n))$ for $n > m > 0$. (For $L = a^*a^n$ n or less agents can guarantee n letters in the derived word, other occurrences of a can be generated by $P_E = \{a \to \lambda, a \to a, a \to aa\}$. Axiom of the system always belongs to the generated language. So eco-grammar system for the singleton contains the only word as axiom and all rules acting in the environment of the eco-grammar system are of the form $a \to a$.) □

Next theorem solves the open problem from Section 3 for unary systems.

Theorem 6.2 $\mathcal{L}(UMEG) = \mathcal{L}(UEG)$.

Proof: $\mathcal{L}(UEG) = \bigcup_{n=1}^{\infty} \mathcal{L}(UEG(n)) = \mathcal{L}(UEG(1)) = \mathcal{L}(UMEG(1)) = \bigcup_{n=1}^{\infty} \mathcal{L}(UMEG(n)) = \mathcal{L}(UMEG)$. □

7 Context-free action rules

In this section the eco-grammar systems with context-free action rules are studied. Such a restriction decreases the generative power of the corresponding eco-grammar systems.

Theorem 7.1 $\mathcal{L}(0EG(n)) \subset \mathcal{L}(EG(n))$ *for* $n \geq 1$,
$\mathcal{L}(X0EG(n)) \subset \mathcal{L}(XEG(n))$ *for* $n \geq 1$ *and* $X \in \{M, U, UM\}$.

Proof: The relation $\subseteq$ follows from the definition in all cases. To prove proper inclusions we consider the language$L_{3,2n} = \{a^{3^i+2n} : i \in N\}$.

$L_{3,2n} \in \mathcal{L}(UMEG(n))$ and $L_{3,2n} = L(\Sigma, a^{3+2n})$ for the simple eco-grammar system $\Sigma = ((\{a\}, \{a \to a^3\}), \{aa \to aa\}^n)$. To prove $L_{3,2n} \notin \mathcal{L}(0EG(n))$ assume contrary that there is an 0EG(n) system $\Sigma' = ((\{a\}, P'_E), A'_1, A'_2, \ldots, A'_n)$ and w' such that $L(\Sigma', w') = L_{3,2n}$.

By Lemma 3.1 we have $P'_E = \{a \to a^{3^k}\}$ for some k. Let n symbols are replaced by agents to at most t symbols in one derivation step. Let us discuss a derivation step $(a^{3^h+2n}, w_1, \ldots, w_n) \Longrightarrow (w', w'_1, \ldots, w'_n)$.

$$|w'| = 3^{h+k} + 3^k n + c$$

where $0 \leq c \leq t-n$ symbols are added by agents. We show that such a system is not able to generate the word $a^{3^{h+1}+2n}$. Assume that $|w'| = 3^{h+k} + 3^k n + c = 3^{h+1} + 2n$ Then we get contradiction $c = 3^h(3-3^k) + n * (2-3^k) < -n < 0 \leq c$. Therefore we conclude with $L_{3,2n} \notin \mathcal{L}(0EG(n))$. □

We compare the power of context-free monocultures with that of context-free eco-grammar systems. We obtain proper inclusion even if no restriction to the number of components is considered. (Compare with open problem in general case.)

Theorem 7.2 *$\mathcal{L}(M0EG) \subset \mathcal{L}(0EG)$ and $\mathcal{L}(UM0EG) \subset \mathcal{L}(U0EG)$.*

Proof: The relation $\subseteq$ follows from the definition in both cases. We prove $L \in \mathcal{L}(0EG) - \mathcal{L}(M0EG)$ for unary language $L = \{a^{4^{2i}} : i \geq 2\} \cup \{a^{4^{4i+1}-4} : i \geq 1\} \cup \{a^{4^{4i+3}-5} : i \geq 1\}$.

We use 0EG system $\Sigma = \{E, A_1, A_2\}$ and the axiom a^{4^4} to generate L.

$$\begin{aligned}
E &= (\{a\}, \{a \to a^4\}),\\
A_1 &= (V, P, R_1, \varphi, \psi_1, A),\\
A_2 &= (V, P, R_2, \varphi, \psi_2, A),\\
V &= \{A, B, C, D\},\\
P &= \{A \to B, B \to C, C \to D, D \to A\},\\
R_1 &= \{a \to \lambda, a \to a^{20}\},\\
R_2 &= \{a \to a^4, a \to a^3, a \to a^8\},\\
\varphi : \varphi(w) &= P,
\end{aligned}$$

$$\begin{aligned}
\psi_1 : \psi_1(A) &= \{a \to \lambda\}, & \psi_2 : \psi_2(A) &= \{a \to a^4\},\\
\psi_1(B) &= \{a \to a^{20}\}, & \psi_2(B) &= \{a \to a^4\},\\
\psi_1(C) &= \{a \to \lambda\}, & \psi_2(C) &= \{a \to a^3\},\\
\psi_1(D) &= \{a \to a^{20}\}, & \psi_2(D) &= \{a \to a^8\}.
\end{aligned}$$

Derivations in Σ for the axiom a^{4^4} are of the form

$(a^{4^4}, A, A) \Longrightarrow (a^{4^5-4}, B, B) \Longrightarrow (a^{4^6}, C, C) \Longrightarrow (a^{4^7-5}, D, D) \Longrightarrow (a^{4^8}, A, A) \Longrightarrow \ldots$
$(a^{4^{4i}}, A, A) \Longrightarrow (a^{4^{4i+1}-4}, B, B) \Longrightarrow (a^{4^{4i+2}}, C, C) \Longrightarrow (a^{4^{4i+3}-5}, D, D) \Longrightarrow (a^{4^{4(i+1)}}, A, A) \Longrightarrow \ldots$

Therefore $L = L(\Sigma, (a^{4^4}, A, A))$.

To prove $L \notin \mathcal{L}(M0EG)$ we proceed by contradiction. Let monoculture $\Sigma' = (E, A^n)$ generates language L. We prove following points i), ..., v) for Σ':

i) $P_E = \{a \to a^{4^l}\}$ for some $l \in N$.

To obtain an "almost" exponential growth of the language we can use only one environmental rule, i.e. $P_E = \{a \to a^{4^l}\}$ for some $l \in N$. (See Lemma 3.1)

ii) All agents use identical action rules in one derivation step.

Let i-th step be the first step where different action rules of agents are used

$(w, w_A^n) \Longrightarrow^{i-1} (w_{i-1}, w_{1,i-1}, w_{2,i-1} \ldots w_{n,i-1}) \Longrightarrow (w_i, w_{1,i}, w_{2,i} \ldots w_{n,i})$.

Suppose that the first and the second agents use different action rules. Denote w_i the word of the environment and let k be the difference between lengths of right sides of action rules used by the first and the second agent. Because all action rules

used in $i-1$ previous steps were the same and all right sides of action rules are the same, similar effect to the environmental word can be obtained after $i-1$ steps by derivations when both agents work like the first or like the second agent, respectively. Follow the $i-$th derivation step in both cases:
$(w, w_A^n) \Longrightarrow^{i-1} (w_{i-1}, w_{1,i-1}, w_{1,i-1}, w_{3,i-1} \dots w_{n,i-1}) \Longrightarrow (w_i', w_{1,i}, w_{1,i}, w_{3,i} \dots w_{n,i})$,
$(w, w_A^n) \Longrightarrow^{i-1} (w_{i-1}, w_{2,i-1}, w_{2,i-1}, w_{3,i-1} \dots w_{n,i-1}) \Longrightarrow (w_i'', w_{2,i}, w_{2,i}, w_{3,i} \dots w_{n,i})$.
We have $|w_i| - |w_i'| = k$ and $|w_i''| - |w_i| = k$, but no three-tuple of words from L fulfills that condition.

Points i) and ii) result to the deterministic behaviour of the system Σ' with respect to the development of the environment.

iii) $w_0 = a^{4^4}$.

Assume that a^{4^4} is not the axiom and follow the derivation step producing word a^{4^4}
$(a^r, w_1, w_2, w_3, \dots, w_n) \Longrightarrow (a^{4^4}, u_1, u_2, \dots u_n)$. Then $r \geq 4^5 - 4$. $P_E = \{a \to a^{4^l}\}$ so action rules of agents eliminate symbols of the environment. Action rules are context-free, i.e. $a \to \lambda$ each component eliminates at most one symbol. To produce a^{4^4} from some $a^r \in L$ we need at least $3 * 4^4 - 4$ components. This blocks the derivation of the word a^{4^4} contrary with the fact that L is infinite. It gives $w_0 = a^{4^4}$.

iv) $l = 1$ and $P_E = \{a \to a^4\}$.

Due to points i)-iii) there is only one possible derivation for the environment in Σ', namely

$$a^{4^4} \Longrightarrow^+ a^{4^5-4} \Longrightarrow^+ a^{4^6} \Longrightarrow^+ a^{4^7-5} \Longrightarrow \dots$$

In this derivation the only rule, $a \to a^{4^l}$, of the environment is used combined by action rules of at most 4^4 agents.
The second part of the above derivation gives

$$4^6 = (4^5 - 4 - n) * 4^l + n * i \geq (4^5 - 4 - n) * 4^l \geq (4^5 - 4 - 4^4) * 4^l \geq (3 * 4^4 - 4) * 4^l$$

This gives $l < 2$.

v) We determine the number of agents n of Σ'. For derivation steps producing a^{4^5-4} and a^{4^7-5} we have

$$\begin{array}{rl} (4^4 - n) * 4 + i * n = 4^5 - 4, & (4^6 - n) * 4 + i * n = 4^7 - 5, \\ n = 4/(4-i), & n = 5/(4-i), \\ 0 \leq i < 4,\ 1 \leq n \leq 4 & i = 3,\ n = 5. \end{array}$$

There is no n satisfying derived conditions. Therefore no M0EG system exists for language L. □

Similarly as in general case we can add to a language of eco-grammar system one special word to obtain a language of context-free monoculture. The construction of the eco-grammar system in the proof of Theorem 4.1 does not save the context-freeness of the action rules. We can add to the alphabet of the environment V_E new letters $u_1, u_2, \dots, u_n$ and use construction analogous to that for general case. In the proof of the next theorem we give another construction which does not increase the size of at least binary environmental alphabet. There are more than two initialization steps needed in this case.

Theorem 7.3 *Let $L \in \mathcal{L}(0EG(n))$ and $|alph(L)| \geq 2$. Then there is a word $u \in (alph(L))^*$ such that $L \cup \{u\} \in \mathcal{L}(M0EG(n))$.*

Proof: The case $L \in \mathcal{L}(M0EG(n))$ is trivial. Let $L \in \mathcal{L}(0EG(n)) - \mathcal{L}(M0EG(n))$ and $L = L(\Sigma, w)$ for axiom w and for $\Sigma = (E, A_1, A_2, \ldots, A_n)$, where $E = (V_E, P_E)$ and $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i, w_{i,0})$, $0 \leq i \leq n$. We choose $u = ba^{n-1}$ for a, b in V_E and we present M0EG(n) system $\Sigma' = (E, A^n)$ for $L \cup \{ba^{n-1}\}$, where axiom is ba^{n-1} and each agent $A = (V, P, R, \varphi, \psi, a)$ is determined by:

$$
\begin{array}{rcl}
V & = & V_1 \cup V_2 \cup \ldots \cup V_n \cup \{a\} \cup \{a_{i,j} : 1 \leq i, j \leq n\} \text{ for} \\
 & & (V_1 \cup V_2 \cup \ldots \cup V_n) \cap (\{a\} \cup \{a_{i,j} : 1 \leq i, j \leq n\}) = \emptyset, \\
P & = & P_1 \cup P_2 \cup \ldots \cup P_n \cup \{a \to a_{i,1} : 1 \leq i \leq n\} \cup \\
 & & \{a_{i,j} \to a_{i,j+1} : 1 \leq i \leq n, 1 \leq j \leq n-1\} \cup \{a_{i,n} \to w_{i,0} : 1 \leq i \leq n\}, \\
R & = & R_1 \cup R_2 \cup \ldots \cup R_n \cup \{b \to b, a \to a, b \to \lambda, a \to w_0\}, \\
\varphi : \varphi(u) & = & \{a \to a_{i,1} : 1 \leq i \leq n\} \cup \\
 & & \{a_{i,j} \to a_{i,j+1} : 1 \leq i \leq n, 1 \leq j \leq n-1\} \cup \{a_{i,n} \to w_{i,0} : 1 \leq i \leq n\}, \\
\varphi(v) & = & \varphi_1(v) \cup \varphi_2(v) \cup \ldots \cup \varphi_n(v) \text{ for } v \in L, \\
\psi : \psi(a) & = & \{a \to a, b \to b\}, \\
\psi(a_{i,i}) & = & \{b \to b\} \text{ for } 1 \leq i \leq n-1, \\
\psi(a_{i,j}) & = & \{a \to a\} \text{ for } i \neq j, 1 \leq i \leq n, 1 \leq j \leq n-1, \\
\psi(a_{n,n}) & = & \{a \to w_0\}, \\
\psi(a_{i,n}) & = & \{b \to \lambda\} \text{ for } 1 \leq i \leq n-1, \\
\psi(x) & = & \psi_i(x) \text{ for all } x \in V_i^*.
\end{array}
$$

We describe the behaviour of the agents in Σ'. All agents start with a. In the first step each agent gets its number as the index i in $a_{i,1}$. Symbol $a_{i,j}$ is the state of i-th agent after j steps of derivation. In the environment an agent in state $a_{i,j}$ rewrites symbol b for $i = j$, otherwise it rewrites symbol a. Two agents can have same number k, but after k steps both of the agents have to rewrite symbol b and there is only one symbol b in the environment, derivation stops. Note that the state of the environment is identical with the axiom during these steps. In a successful derivation we have one agent of each type after n steps. The last agent rewrites the symbol b to w_0 and the other agents eliminate a-s. In the same time the states of all agents are rewritten to their original starting state in Σ. Formally

$$
\begin{array}{l}
(ba^{n-1}, a^n) \Longrightarrow (ba^{n-1}, a_{1,1}, a_{2,1}, \ldots, a_{n,1}) \Longrightarrow (ba^{n-1}, a_{1,2}, a_{2,2}, \ldots, a_{n,2}) \Longrightarrow \ldots \Longrightarrow \\
(ba^{n-1}, a_{1,n-1}, a_{2,n-1}, \ldots, a_{n,n-1}) \Longrightarrow (ba^{n-1}, a_{1,n}, a_{2,n}, \ldots, a_{n,n}) \Longrightarrow \\
(w_0, w_{1,0}, w_{2,0}, \ldots, w_{n,0})
\end{array}
$$

Σ' is in starting configuration of system Σ and next steps follow derivation in Σ, therefore $L(\Sigma', ba^{n-1}) = L(\Sigma, w) \cup \{ba^{n-1}\}$. □

We continue with comparison of the generative power of monocultures with different number of components. The restriction to context-free action rules destroys the (partial) hierarchy on language classes introduced by the number of components of systems.

Theorem 7.4 *$\mathcal{L}(X0EG(n))$ and $\mathcal{L}(X0EG(m))$ are incomparable but not disjoint for $X \in \{M, U, UM\}$ and $n, m \in N, n \neq m$.*

Proof: Let $n > m$. $\{a^m, \lambda\} \in \mathcal{L}(UM0EG(m)) - \mathcal{L}(0EG(n))$ by Lemma 3.2.

It is enough to prove $L_{n+1,n+1} \in \mathcal{L}(UM0EG(n)) - \mathcal{L}(0EG(m))$ where $L_{n+1,n+1} = \{a^{(n+1)^i+n+1} : i \in N\}$. We have $L_{n+1,n+1} = L(\Sigma, a^{2n+2})$ for the simple unary M0EG(n) system $\Sigma = ((\{a\}, \{a \to a^{n+1}\}), \{a \to \lambda\}^n)$.

To verify $L_{n+1,n+1} \notin \mathcal{L}(0EG(m))$ for $n > m$ assume contrary that some 0EG(m) system $\Sigma' = ((\{a\}, P_E), A'_1, A'_2, \ldots, A'_m)$ generates $L_{n+1,n+1}$. By Lemma 3.1 we have $P'_E = \{a \to a^{(n+1)^k}\}$ for some k.

Consider a derivation step $(a^{(n+1)^j+n+1}, w_1, w_2, \ldots, w_n) \Longrightarrow (w', w'_1, w'_2, \ldots, w'_n)$ For $n > m$ at least $(n+1)^j + 2$ symbols of the environment are rewritten by the rule of the environment $a \to a^{(n+1)^k}$ to $(n+1)^{j+k} + 2(n+1)^k$ symbols. Agents can add some symbols $c \geq 0$ up to agents maximum c^A.

$(n+1)^{j+k} + 2(n+1)^k + c \leq (n+1)^{j+r} + (n+1)$ for $r \geq k$,

$c \leq (n+1)^{j+r} - (n+1)^{j+k} - 2(n+1)^k + (n+1) < 0$.

System Σ' does not generate $L_{n+1,n+1}$ and there is no U0EG(m) system for $L_{n+1,n+1}$. □

References

[1] M. H. ter Beek. Simple eco-grammar systems with prescribed teams. In: *Grammatical models of multi-agent systems*, G. Păun, A. Salomaa (eds.), Gordon and Breach, Yverdon, 1999, 113–135

[2] J. Csima. On extended simple eco-grammar systems. *Acta Cybernetica 13* (1998) 359–373

[3] J. Csima. On hybrid eco-rewriting systems. *Grammars 1* (1999) 239–253

[4] J. Csima. Two remarks on variants of simple eco-grammar systems. *Acta Cybernetica 14* (2000) 569–582

[5] E. Csuhaj-Varjú. Eco-grammar systems: Recent results and perspectives. In: *Artificial Life: Grammatical models*, ed. G. Păun, Black Sea University Press, 1995, 79–103

[6] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, G. Păun. Eco(grammar) systems. A preview. In: *Proc. 12th European Meeting on Cybernetics and System Research, Vienna, 1994,* ed. R. Trappl, World Scientific, Singapore, 1994, 941–948

[7] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, G. Păun. Eco-grammar systems. A grammatical framework for studying lifelike interactions. *Artificial Life 3(1)* (1997) 1–28

[8] E. Csuhaj-Varjú, A. Kelemenová. Team behavior in eco-grammar systems. *Theoretical computer science 209* (1998) 213–224

[9] E. Csuhaj-Varjú, V. Mitrana. Dynamical teams in eco-grammar systems. *Fundamenta Informaticae 44* (2000) 83–94

[10] E. Csuhaj-Varjú, G. Păun, A. Salomaa. Conditional tabled eco-grammar systems versus (E)TOL systems. *Journal of Universal Computer Science 5(1)* (1995) 252–269

[11] A. Kelemenová. Descriptional complexity of eco-grammar systems. In: *Proc. Workshop DCFS 2003, Descriptional complexity of formal systems,* E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszil (eds.), MTA SZTAKI Budapest, 2003, 42–52

[12] A. Kelemenová. Eco-grammar systems. In: *Formal languages and applications,* Physica Verlag, Heidelberg, 11 pp. (in print)

[13] A. Kelemenová, J. Kelemen. From colonies to eco(grammar) systems. An overview. In: *Proc. Important Results and Trends in Theoretical Computer Science, Graz, 1994.* J. Karhumäki, H. A. Maurer, G. Rozenberg (eds.), Lecture Notes in Computer Science 812, Springer Verlag, Berlin, 1994, 211–231

[14] G. Păun. Eco-grammar systems: An early survey. In: *Decentralized intelligent multi-agent systems symposium.* Cracow, 1995, 355–362

[15] P. Sosík. Eco-grammar systems, decidability and the tiling problem. In: *Proc. MFCS 98 satellite workshop Grammar systems*, ed, A. Kelemenová, Silesian University, Opava, 1998, 195–213

[16] P. Sosík. On the decidability problems of eco-grammar systems. *Journal of Automata, Languages and Combinatorics* (2000) 1 45–58

[17] D. Watjen. Remarks on simple eco-grammar systems with prescribed teams. *Journal of Automata, Languages and Combinatorics 8* (2003) 1 85–108

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 255 - 275, 2004.

Grammar Systems vs. Membrane Computing: A Preliminary Approach

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro
and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Abstract

Grammar systems and membrane computing are two well developed branches of (theoretical) computer science, having many things in common, but whose relationships and cross-fertilization possibilities were not yet systematically investigated. This paper starts such a systematic study, trying to import features of (mathematical or computational) interest from one area to another one. We consider in some detail the case of cooperating distributed (CD) and parallel communicating (PC) grammar systems: the t-mode of cooperation from CD grammar systems can be used instead of the target indications from cell-like P systems, while the use of multisets of strings in PC grammar systems leads to a sort of tissue-like P systems (able to solve SAT in linear time). The paper has a preliminary character; many open problems and research topics are formulated.

1 Introduction

Grammar systems and membrane computing are two active areas of theoretical computer science, with different starting points and motivations, but with several similarities (both areas deal with distributed computing devices, where such notions as parallelism, cooperation, communication, decentralization are crucial).

The basic idea of grammar systems is to have several grammars cooperating together according to a specified protocol in generating a unique language. The initial motivations were related to two-level grammars and artificial intelligence issues (we refer to [1] for details), but ideas from robotics, eco-systems, internet, distributed and parallel computing were later incorporated. Two main classes of systems were initially considered: *cooperating distributed* (CD) grammar systems, with the component grammars working sequentially, in turns, on a common sentential form, and

parallel communicating (PC) grammar systems, where each component grammar works on its own sentential form, and communicates on request with other components. In the case of CD grammar systems, five basic cooperation protocols are investigated, stating when a component can/must leave the sentential form to be processed by the other components: at any time (the mode $*$), after at most, at least, or a specified number of steps (the modes $\leq k, \geq k, = k$, respectively, for a specified k), when no further rule can be used (the mode t). This latest mode will be considered here, using it in order to control the communication among regions of cell-like membrane systems.

In its turn, membrane computing starts from the goal of defining a computing model inspired from the structure and the functioning of the living cell, hence it is a branch of *natural computing*, a powerful trend now in computer science. In short, in the compartments of a hierarchical arrangement of membranes one places *objects* which evolve (in a maximally parallel manner) by means of rules which are also localized in regions. In the basic variant of such systems (called P systems) one works with symbol-objects, hence the evolution rules are multiset-processing rules, but here we consider the case where the objects are structured and can be described by strings over a given alphabet. Such P systems, with the objects processed by usual context-free rewriting rules, were already considered in the literature (we refer to [11] for details). In these systems, the string-objects are passed from a compartment to another one according to target commands associated with rewriting rules; specifically, the rules are of the form $X \to u(tar)$, where $X \to u$ is a usual context-free rule and tar is one of the indications *here, in, out*, with the meaning that the string obtained after the use of the rule $X \to u$ remains in the same region, goes into a directly lower region, or exits the membrane, going to the surrounding region, respectively.

Comparing the CD grammar systems and P systems, one can easily see the similarities and the differences. It is also obvious that ideas from one type of systems can be used in the functioning of the other type. Actually, the question of defining the communication of strings among the regions of a P system based on principles as used in CD grammar systems was formulated several times. In a different context, P systems (with symbol-objects) with limited parallelism, reminding the modes $\leq k$ and $= k$, were already investigated in [5], [7]. The case $= 1$ corresponds to P systems with immediate communication (see [11]). In what follows, we investigate the case of mode t (and we leave as a topic of research for the reader to consider the modes $*, \leq k, = k, \geq k$ for the case of string-objects P systems).

More exactly, we replace the target indication in, or the target indication out, or both, by t: a string is processed in a given region until no further rule can be applied to it in that region; then, it will go to an inner region, to the surrounding region, or to any of these regions. We identify these cases with $tin, tout, tgo$, respectively. (In the first two cases, the commands out, in, respectively, remain to be introduced by rules, as usual in string-object P systems.) The power of P systems with these types of communication commands is investigated, in comparison with the power of CD grammar systems, of Chomsky grammars, and of Lindenmayer systems. The results are as expected: universality for a small number of membranes, covering CF and $ET0L$ families at certain levels of the obtained hierarchies – without however

knowing whether or not these results are optimal.

In what concerns the PC grammar systems, their natural membrane computing counterpart are the tissue-like P systems, where the membranes are placed in the nodes of a graph, and they communicate along a given set of channels (edges in the graph). A P system uses however multisets of strings and communicates them by means of target indications associated with the rewriting rules. In PC grammar systems, one communicates on request, by means of query symbols. Thus, an immediate idea is to consider PC grammar systems with multisets of strings in each component, processed simultaneously; this can be also interpreted as a P system with the communication done on request: when a component introduces one or more query symbols in a string, then the rewriting of that string stops and the queries are satisfied by replacing the query symbols by *all* strings which do not contain query symbols (if no such a string exists in the queried component, then the string containing a query symbol which cannot be satisfied disappears – it is like replacing this query symbol by elements of an empty set) from the component indicated by the query symbol, in all possible combinations (if, for instance, in a string one introduces two query symbols, pointing to two components with two available strings each, then we get in total four strings in the receiving component). In this way, the strings can be replicated, which suggests that such systems can solve computationally hard problems in an efficient way. We confirm this expectation by proving that `SAT` can be solved in this framework in a polynomial time.

We do not investigate here the power of multiset PC grammar systems but we only show that systems with one component can already generate non-context-free languages. It is highly probable that these systems are computationally universal, they can characterize the family of recursively enumerable languages. Finding how many components would be sufficient to this aim remains as an open problem.

Actually, several open problems and research topics are formulated below, and, in some sense, this should be considered the main contribution of this preliminary version of the paper: calling the attention to the fruitfulness of bridging grammar systems area with membrane computing area.

2 Prerequisites

We introduce here some notation and terminology, but the reader is assumed to have some familiarity with basic elements of formal language theory (regulated rewriting included), grammar systems, and membrane computing, e.g., from the monographs mentioned in the bibliography.

Our notations are as follows: V^* is the free monoid generated by the alphabet V with respect to the operation of concatenation; λ is the empty string; the length of $x \in V^*$ is denoted by $|x|$, and $|x|_U$ is the number of occurrences in x of symbols from $U \subseteq V$; a Chomsky grammar is presented in the form $G = (N, T, S, P)$, where N is the non-terminal alphabet, T is the terminal alphabet, S is the axiom, and P is the set of rewriting rules; LIN, CF, CS, RE are the families of linear, context-free, context-sensitive, recursively enumerable languages, respectively; $ET0L$ is the family of languages generated by extended tabled interactionless Lindenmayer (ET0L)

systems.

In the universality proofs we will use the notion of a matrix grammar (with appearance checking) in the strong binary normal form. Such a grammar will be given as a tuple $G = (N, T, S, M, F)$, where $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and with the matrices in M in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$.

Moreover, there is only one matrix of type 1 (that is why one uses to write it in the form $(S \to X_0A_0)$, in order to fix the symbols X, A present in it), and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; there are at most two symbols $A \in N_2$ which appear in rules of the form $A \to \#$ (we identify these symbols with $B^{(1)}$ and $B^{(2)}$); $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar there is an equivalent matrix grammar in the strong binary normal form, hence such grammars characterize RE.

For the sake of completeness, we give here some basic definitions from grammar systems area.

A CD grammar system is a construct of the form $\Gamma = (N, T, S, P_1, \ldots, P_m)$, where N, T are disjoint alphabets, $S \in N$, and $P_1, \ldots, P_m$ are finite sets of context-free rules over $N \cup T$ (with N interpreted as the non-terminal alphabet and T the terminal alphabet of the system). The sets P_i are called *components* of Γ, and their number – m above – is the *degree* of the system.

The derivation relation according to a set P_i is defined as usual in the context-free grammar $G_i = (N, T, S, P_i)$ and denoted by $\Rightarrow_i$, $1 \le i \le n$. Then, for $x, y \in (N \cup T)^*$ we define

$$x \Rightarrow_i^t y \quad \text{iff} \quad x \Rightarrow_i^* y \text{ and there is no } z \in (N \cup T)^* \text{ such that } y \Rightarrow_i z.$$

The language generated by Γ *in the t-mode of cooperation* is

$$\begin{aligned} L_t(\Gamma) \quad = \quad & \{w \in T^* \mid S \Rightarrow_{i_1}^t w_1 \Rightarrow_{i_2}^t \ldots \Rightarrow_{i_k}^t w_k = w, \\ & \text{for some } 1 \le i_1, i_2, \ldots, i_k \le m,\ k \ge 1\}. \end{aligned}$$

That is, the sentential form is processed by a component until no rule of that component can be applied; then, another component, non-deterministically chosen, takes the string and rewrites it; we start from the axiom S and we collect in the generated language all strings over T.

The family of languages generated by CD grammar systems of degree at most $m \ge 1$ in the t-mode of cooperation is denoted by $CD_m(t)$; if the number of components is not restricted, then the corresponding family (hence the union of all families $CD_m(t), m \ge 1$) is denoted by $CD_*(t)$. The following relations are know:

$$CF = CD_1(t) = CD_2(t) \subset CD_3(t) = CD_*(t) = ET0L.$$

That is, systems with one or two components characterize the context-free languages, while three components already give the maximal power of such systems, namely, a characterization of ET0L languages.

It is interesting to note that if the order of enabling the components of a CD grammar system is controlled by a graph, then the power is not increased, still we characterize $ET0L$ (by systems with three components).

We introduce now the standard PC grammar systems (with one string in each component), which are constructs of the form $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_m, P_m))$, $m \geq 1$, where N, T, and K are pairwise disjoint alphabets (of non-terminal symbols, terminal symbols, and query symbols, respectively), and for all $1 \leq i \leq m$, $S_i \in N$ (axioms) and P_i are finite sets of context-free rewriting rules of the form $A \to u$, with $A \in N, u \in (N \cup T \cup K)^*$. Each $(S_i, P_i), 1 \leq i \leq m$, is called *a component* of Γ. One of the components is said to be the *master* of Γ. Without any loss of generality, this can be the first one, (S_1, P_1).

For a system as above, an m-tuple $(x_1, \ldots, x_m)$ with $x_i \in (N \cup T \cup K)^*$, $1 \leq i \leq m$, is called a *configuration* of Γ. $(S_1, \ldots, S_m)$ is the *initial configuration.*

PC grammar systems change their configurations by performing direct derivation steps, in the following way. Let $(x_1, \ldots, x_m)$ and $(y_1, \ldots, y_m)$ be two configurations of a system Γ. We say that $(x_1, \ldots, x_m)$ *directly derives* $(y_1, \ldots, y_m)$ (in modeα, to be specified below), denoted by $(x_1, \ldots, x_m) \Rightarrow_\alpha (y_1, \ldots, y_m)$, if one of the following two cases holds:

1. There is no x_i which contains any query symbol, that is, $x_i \in (N \cup T)^*$ for all $1 \leq i \leq m$. Then, for each i, $1 \leq i \leq m$, $x_i \Rightarrow_i y_i$ (y_i is obtained from x_i by a direct derivation step in component i) for $x_i \notin T^*$, and $x_i = y_i$ for $x_i \in T^*$.

2. There is some x_i, $1 \leq i \leq m$, which contains at least one occurrence of a query symbol. Then, for each x_i, $1 \leq i \leq m$, with $|x_i|_K \neq 0$ we write $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, where $z_j \in (N \cup T)^*$, $1 \leq j \leq t+1$, and $Q_{i_l} \in K$, $1 \leq l \leq t$. If $|x_{i_l}|_K = 0$ for each l, $1 \leq l \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ and (a) in *returning* systems we have $y_{i_l} = S_{i_l}$, while (b) in *non-returning* systems we have $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$. If $|x_{i_l}|_K \neq 0$ for some l, $1 \leq l \leq t$, then $y_i = x_i$. For all j, $1 \leq j \leq m$, for which y_j is not specified above, $y_j = x_j$.

In the returning mode, $\alpha = R$, while in the non-returning mode we write $\alpha = nR$.

Let $\Rightarrow_\alpha^*$ denote the reflexive and transitive closure of $\Rightarrow_\alpha, \alpha \in \{R, nR\}$. Then, the language generated by the system Γ (with the master component (S_1, P_1)) in the mode α is $L_\alpha(\Gamma) = \{x \in T^* \mid (S_1, \ldots, S_m) \Rightarrow_\alpha^* (x_1, \ldots, x_m)$, for some $x_1, \ldots, x_n \in (N \cup T \cup K)^*$ such that $x = x_1\}$.

Let the class of languages generated by returning PC grammar systems having at most m context-free components be denoted by RPC_mCF and the corresponding family of languages generated in the non-returning mode be denoted by $nRPC_mCF$. When the number of components is not limited, we replace the subscript m with $*$.

The following relations are know (see [11], [3], [9]):

1. $CF = RPC_1CF = nRPC_1CF \subset (RPC_2CF \cap nRPC_2CF)$.

2. $RE = RPC_5CF = nRPC_*CF$.

That is, systems with one component generate only context-free languages, two components suffice for generating non-context-free languages, while returning systems

with 5 components and non-returning systems (with no bound on the number of components) characterize the recursively enumerable languages.

3 From CD to P: P Systems Using the t-Communication

We pass now to exploring the possibility to use features of grammar systems in P systems, and conversely. In this section we consider the way from CD to P, namely, we use the t-mode of cooperating in a CD grammar system as a substitute for target indications *in, out* in a P system.

We introduce the classes of P systems we are going to investigate, for all cases where the t-mode can be used as a communication mode, then we illustrate the definition with some examples; results about the power of the obtained systems are given (in most cases, without a proof) in a diagrammatic form (it is highly possible that several of these results can be improved).

An extended P system (of degree $m \geq 1$) with string-objects and *t-communication* is a construct

$$\Pi = (V, T, \mu, (w, i_0), R_1, \ldots, R_m),$$

where:

1. V, T are alphabets such that $T \subseteq V$;
2. μ is a *membrane structure* (of degree m, with the membranes labeled in a one-to-one manner with elements of a set H; in this definition we use $H = \{1, 2, \ldots, m\}$);
3. w is a non-empty string over V, present in region i_0 of μ, for some $1 \leq i_0 \leq m$;
4. $R_1, \ldots, R_m$ are finite sets of *evolution rules* associated with the m regions (membranes) of μ; in what follows, the rules are of the forms $a \to u$ or $a \to u(tar)$, where $a \in V$, $u \in V^*$, and $tar \in \{in, out\}$. In a given system at most one of the target indications in, out may be present in the rules.

We note the difference between the definition above and that of usual P systems with string-objects, where several axiom-strings are present in the system at the beginning of any computation. Because we do not consider here halting computations, the evolution of each string is independent from the evolution of other strings; also in order to be closer to the form of a CD grammar system, we consider here only one string initially present in the system. This means that in any moment of a computation there is only one string in the system, which is either eventually sent into the environment, or remains forever inside. This string is rewritten by the rules from the region where it is placed; in each step, only one rule is applied (hence the rewriting is performed in a sequential manner); if a string cannot be rewritten, then it remains unchanged.

New here is also the way the strings are communicated among regions. If a string is rewritten by a rule $a \to u(in)$, then the string obtained after rewriting is immediately moved to one of the directly inner regions, non-deterministically chosen;

if no inner membrane exists, then such a rule cannot be applied. If the used rule was $a \to u(out)$, then the resulting string is immediately sent out of the membrane where it was produced. If we use a rule without any target indication, then the resulting string remains in the same region if it can be further rewritten there, or it exits if no local rule can be applied to it, being communicated to one of the adjacent regions as specified below.

Three cases (three types of systems) are distinguished:

1. A system which uses rules of the form $a \to u(in)$ (hence not also of type $a \to u(out)$) is said to be of *tout* type; a string which cannot be further rewritten in a given region is communicated to the upper region, that is, the t-mode from CD grammar systems enforces the *out* target command.

2. A system which uses rules of the form $a \to u(out)$ (hence not also of type $a \to u(in)$ is said to be of *tin* type; a string which cannot be further rewritten in a given region is communicated to one of the directly lower regions, that is, the t-mode from CD grammar systems enforces the *in* target command; if the membrane is elementary, then the string remains forever in that region.

3. A system which uses only rules of the form $a \to u$ (hence without any target command) is said to be of *tgo* type; a string which cannot be further rewritten in a given region is communicated either to the upper region or to one of the directly lower regions, non-deterministically choosing the direction and the region.

In all cases, if a string arrives in a region where no rule can be applied to it, then this is interpreted as a maximal derivation in that region and the string is moved immediately up or down in the membrane structure, according to the type of the system.

By using the rules and moving the strings as specified above, we get a computation. The language generated by the system consists of all strings over T which are sent out of the system during any possible computation. We note again that we do not work here with halting computations – a string sent into the environment cannot be further processed, hence its evolution is finished.

The language generated by a system Π is denoted by $L(\Pi)$. The family of all languages generated by systems of degrees at most $m \geq 1$ of type $\alpha \in \{tout, tin, tgo\}$ is denoted by $ELSP_m(\alpha)$; if we use systems of an arbitrary degree, then we replace the subscript m with $*$; when we use *non-extended* systems, that is with $T = V$, the front letter E is removed.

We illustrate the previous definitions with two simple **examples**. Consider the system – of type *tout*:

$$\begin{aligned}
\Pi &= (V, T, \mu, (w, i_0), R_1, R_2), \text{ where} \\
V &= \{a, b, c, c', d, d'\}, \\
T &= \{a, b\}, \\
\mu &= [_1 [_2 \,]_2]_1, \\
w &= cd, \ i_0 = 1,
\end{aligned}$$

$$\begin{aligned} R_1 &= \{c \to ac'b(in),\ d \to ad'b,\ d' \to d', \\ &\quad\ c \to ab,\ d \to ab\}, \\ R_2 &= \{c' \to c,\ d' \to d\}. \end{aligned}$$

Assume that we have a string $a^n cb^n a^m db^m$ in region 1, with some $n \geq m \geq 0$; initially, $n = m = 0$. If we apply the rule $c \to ac'b(in)$, then the string goes immediately to region 2; if we apply first the rule $d \to ad'b$, then the string remains in region 1, where it can evolve forever by means of rule $d' \to d'$, hence, in order to proceed further, we have to also use the rule $c \to ac'b(in)$. Thus, we send to region 2 a string of the form $a^{n+1}c'b^{n+1}a^m db^m\ a^{n+1}c'b^{n+1}a^{m+1}d'b^{m+1}$. In region 2 we have to perform a maximal derivation, hence we return to region 1 a string $a^{n+1}cb^{n+1}a^{m+i}db^{m+i}$, for $i \in \{0, 1\}$. The process is iterated. If we use the rules $c \to ab,\ d \to ab$ from region 1, then we send out a terminal string. If only the first of these rules is used, then the computation will produce no result, because we obtain a string of the form $a^n cb^n a^m d'b^m$ which cannot leave region 1. If we use only the rule $d \to ab$, then the computation will continue by increasing the number of occurrences of a and b in the prefix $a^n b^n$ of the string. Therefore, $L(\Pi) = \{a^n b^n a^m b^m \mid n \geq m \geq 1\}$. Note that this is not a context-free language and that the generated language is the same if we consider $T = V$ (the non-extended counterpart of the system): as long as any symbol from $V - T$ is present, the string can be rewritten, hence it cannot leave the system.

A system which generates a non-context-free language can be easily constructed also for the *tin* case. We consider the system

$$\begin{aligned} \Pi &= (\{a, b, b', c, c'\}, \{a, b, c\}, [_1[_2\ ]_2]_1, (bc, 1), R_1, R_2), \text{ where} \\ R_1 &= \{b \to ab'a,\ c \to ac'a,\ c'' \to aca(out)\}, \\ R_2 &= \{b' \to b,\ c' \to c(out),\ c' \to c''(out)\}. \end{aligned}$$

The string is again repeatedly moved across membrane 2, increasing either both "blocks" $a^n b^n$ or only the second one (this latter case happens when in region 2 we do not use the rule $b' \to b$ but only the rule $c' \to c(out)$). When sent out, because c' is not a terminal symbol, the use of the rule $c'' \to aca(out)$ ensures the increase of the second "block", hence the generated language is $L(\Pi) = \{a^n ba^n a^m ca^m \mid m > n \geq 1\}$.

Thus, the families $LSP_2(tout), LSP_2(tin)$ contain non-context-free languages (passing from the language generated by the non-extended version of a system to the extended one corresponds to an intersection with T^*, and CF is closed under intersection with regular languages). This result can be strengthened: these families also contain languages which are not semilinear, but we omit the proof of this assertion.

4 The Power of P Systems of *tout* and *tin* Types

The relationships from the diagram from Figure 1 hold (the arrows indicate inclusions which are not known to be proper, while the arrows marked with a dot indicate strict inclusions; the unrelated families are not necessarily incomparable).

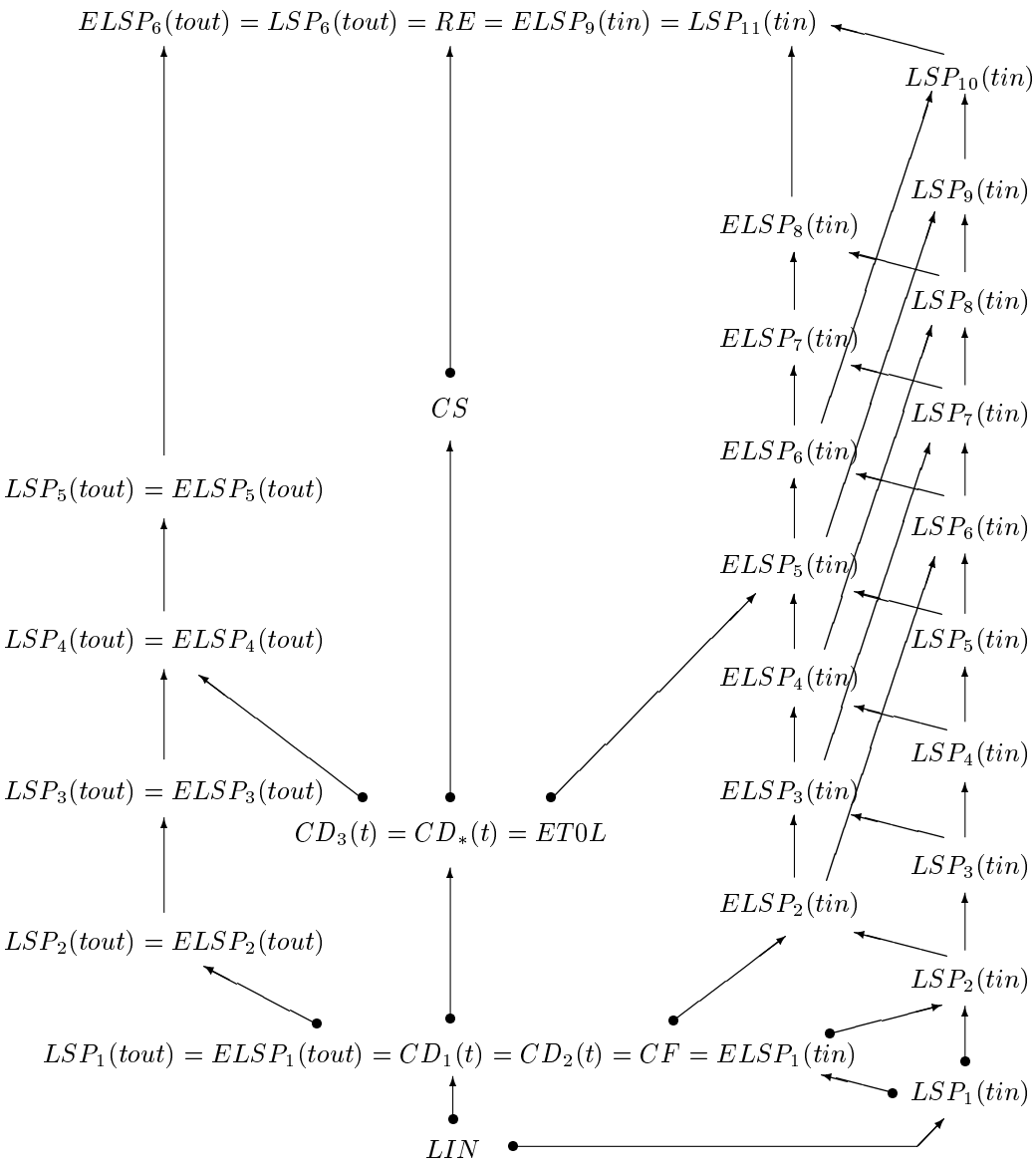
$ELSP_6(tout) = LSP_6(tout) = RE = ELSP_9(tin) = LSP_{11}(tin)$
$LSP_{10}(tin)$
$LSP_9(tin)$
$ELSP_8(tin)$
$LSP_8(tin)$
$ELSP_7(tin)$
CS
$LSP_7(tin)$
$ELSP_6(tin)$
$LSP_5(tout) = ELSP_5(tout)$
$LSP_6(tin)$
$ELSP_5(tin)$
$LSP_4(tout) = ELSP_4(tout)$
$LSP_5(tin)$
$ELSP_4(tin)$
$LSP_4(tin)$
$LSP_3(tout) = ELSP_3(tout)$
$ELSP_3(tin)$
$CD_3(t) = CD_*(t) = ET0L$
$LSP_3(tin)$
$ELSP_2(tin)$
$LSP_2(tout) = ELSP_2(tout)$
$LSP_2(tin)$
$LSP_1(tout) = ELSP_1(tout) = CD_1(t) = CD_2(t) = CF = ELSP_1(tin)$
$LSP_1(tin)$
LIN

Figure 1.

We give here the proofs for two of the relations from this diagram, namely, for the universality results.

Lemma 4.1 $ELSP_6(tout) = RE$.

Proof. We start from a matrix grammar in the strong binary normal form, $G = (N, T, S, M, F)$, with k matrices of the form $m_i : (X \to Y, A \to u)$ and h_j matrices of the form $m_i : (X \to Y, B^{(j)} \to \#)$, $j = 1, 2$. In all terminal matrices $(X \to \lambda, A \to u)$ we replace the first rule with $X \to f$, where f is a new symbol.

We construct the P system of degree 6

$$\begin{aligned}
\Pi &= (V, V, \mu, (X_0A_0, 1), R_1, \ldots, R_6), \text{ where} \\
V &= \{X, X', X'', X''' \mid X \in N_1 \cup \{f\}\} \cup T \\
&\cup \{X_{i,j}, X'_{i,j}, X''_{i,j}, X'''_{i,j}, \bar{X}_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
&\cup \{A_{i,j}, A'_{i,j}, A''_{i,j}, A'''_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
&\cup \{X_i^{(j)} \mid m_i : (X \to Y, B^{(j)} \to \#), 1 \le i \le k_j, j = 1, 2\}, \\
\mu &= [_1[_2[_3[_4\ ]_4]_3]_2[_5\ ]_5[_6\ ]_6]_1,
\end{aligned}$$

and with the sets of rules given in Table 1 (in all cases, $1 \le j < i$ and $s = 1, 2$):

We also add the following rules to set R_1:

$$\begin{aligned}
&X \to X,\ X \in N_1, \\
&A \to A,\ A \in N_2, \\
&f''' \to \lambda.
\end{aligned}$$

Membranes 5 and 6 are used for simulating the matrices $m_i : (X \to Y, B^{(s)} \to \#)$, for $s = 1, 2$, respectively, while membranes 2, 3, 4, together with the skin region, simulate the matrices of the form $m_i : (X \to Y, A \to u)$, $1 \le i \le k$.

Specifically, the string Xw from the skin region is sent to one of the directly inner membranes by using one of the rules $A \to A_{i,0}(in)$, $X \to X_i^{(s)}(in)$. If the string is sent to a "wrong" membrane, then it never leaves that membrane, because of the rules of the form $\alpha \to \alpha$.

If we want to simulate a matrix $m_i : (X \to Y, A \to u)$, hence the string has the form $Xw_1A_{i,0}w_2$ and it has arrived in membrane 2, then we continue by using a rule $X \to \bar{X}_{r,0}(in)$ and the string $\bar{X}_{r,0}w_1A_{i,0}w_2$ is sent to membrane 3. From here, it is sent (with the bar of X removed) to membrane 4, where the second subscript of symbols X and A is increased by one. Assume that we are in a stage when a string $X'_{r,r'}w_1A'_{i,i'}w_2$ is sent from region 4 to region 3. The string can exit only after having both X and A double primed, then from membrane 2 it is sent to the skin region, with A being triple primed. Now, in the skin region we can use either the rule $X''_{r,r'} \to X'''_{r,r'}(in)$, and this is the correct continuation, or a rule of the form $B \to B_{t,0}(in)$. In the latter case, the string will arrive in region 2 and will remain forever here, rewritten by the rule $B_{t,0} \to B_{t,0}$. In the former case, the string is sent to region 3 by means of the rule $X'''_{r,r'} \to X_{i,j}(in)$ from R_2, and then to region 4 by means of the rule $A'''_{i,i'} \to A_{i,i'}(in)$ from R_3. In this way, the process of increasing the second subscript of the two symbols X and A is iterated.

Table 1: The rules of the system from the proof of Lemma 4.1

	$m_i : (X \to Y, A \to u)$	$m_i : (X \to Y, B^{(s)} \to \#)$
R_1	$A \to A_{i,0}(in)$ $X''_{i,j} \to X'''_{i,j}(in)$ $X_{i,j} \to X_{i,j}(in)$ $Y''' \to Y$	$X \to X_i^{(s)}(in)$
R_2	$X \to \bar{X}_{i,0}(in)$ $X'''_{i,j} \to X_{i,j}(in)$ $A''_{i,j} \to A'''_{i,j}$ $A_{i,0} \to A_{i,0}$ $Y'' \to Y'''$	$X_i^{(s)} \to X_i^{(s)}$
R_3	$\bar{X}_{i,0} \to X_{i,0}(in)$ $X'_{i,j} \to X''_{i,j}$ $A'_{i,j} \to A''_{i,j}$ $A'''_{i,j} \to A_{i,j}(in)$ $Y' \to Y''$	
R_4	$X_{i,j} \to X_{i,j+1}$ $A_{i,j} \to A_{i,j+1}$ $X_{i,i} \to Y'$ $A_{i,i} \to u$ $\bar{X}_{i,0} \to \bar{X}_{i,0}$ $A'''_{i,j} \to A'''_{i,j}$	
R_5	$X_{i,0} \to X_{i,0}$ $A_{i,0} \to A_{i,0}$ $X'''_{i,j} \to X'''_{i,j}$	$X_i^{(1)} \to Y$ $X_i^{(2)} \to X_i^{(2)}$ $B^{(1)} \to B^{(1)}$
R_6	$X_{i,0} \to X_{i,0}$ $A_{i,0} \to A_{i,0}$ $X'''_{i,j} \to X'''_{i,j}$	$X_i^{(2)} \to Y$ $X_i^{(1)} \to X_i^{(1)}$ $B^{(2)} \to B^{(2)}$

If both the symbols X and A get a subscript i, i, then the matrix m_i is simulated.

Assume that only X gets identical subscripts, hence we send from membrane 4 to membrane 3 a string $Y' w_1 A'_{i,i'} w_2$; we pass to $Y'' w_1 A''_{i,i'} w_2$, which is sent to region 2, where we produce $Y''' w_1 A'''_{i,i'} w_2$ which is sent to the skin region. The string cannot exit the system, because of the rules $Y''' \to Y$, $Y \to Y$, hence in order to continue we have to apply a rule $B \to B_{t,0}(in)$. If Y is present (hence the rule $Y''' \to Y$ was used), then we can move the string to membrane 3, otherwise we get stuck, the rule $B_{t,0} \to B_{t,0}$ can be used forever. If the string $\bar{Y}_{s,0} w_1 A'''_{i,i'} w_2$ is present in membrane 3, then is will be sent to membrane 4 by using one of the rules $\bar{Y}_{s,0} \to Y_{s,0}(in)$ (and then $A'''_{i,i'}$ is still present) and $A'''_{i,j} \to A_{i,j}(in)$ (and then $\bar{X}_{s,0}$ will be present). In either case, the string remains forever in membrane 4.

Assume now that we use the rule $A_{i,i} \to u$ from R_4 without also using the rule $X_{i,i} \to Y'$, hence we send to membrane 3 a string $X'_{r,r'} w$. It will pass to membrane

2 in the form $X''_{r,r'}w$ and from here it will go unchanged to the skin region. If we apply here a rule $B \to B_{s,0}(in)$, then the string will get stuck in region 2. If we apply the rule $X''_{r,r'} \to X'''_{r,r'}(in)$ from R_1, then the string returns to region 2, here the symbol X loses the primes and the string is sent to region 3, where nothing can happen. The string is sent out to region 2, and from here to region 1, unchanged, but it comes back because of the rule $X_{r,r'} \to X_{r,r'}(in)$. If a rule $B \to B_{s,0}(in)$ is used, then the string arrives in region 2 and it never exits.

Therefore, the simulation of the matrix m_i should be complete, otherwise we get no result.

If the matrix was terminal, then the string should be terminal, otherwise it is sent to region 2 and gets stuck there, because of the rules $A_{i,0} \to A_{i,0}$.

If we want to simulate a matrix $m_i : (X \to Y, B^{(j)} \to \#)$, hence a rule $X \to X_i^{(j)}$ is used, then either the string gets stuck in membranes 2 or $7-j$, or it arrives in the right membrane $4+j$. If the symbol $B^{(j)}$ is present, then the string remains forever in this membrane, otherwise it can exit, after using the rule $X_i^{(j)} \to Y$, which correctly simulates the matrix.

Because of the rules $A \to A, X \to X$ from the skin region, only strings which are terminal with respect to G can be sent out. Consequently, $L(G) = L(\Pi)$. □

Lemma 4.2 $ELSP_9(tin) = LSP_{11}(tin) = RE$.

Proof. Consider first the extended case. In order to prove the inclusion $RE \subseteq ELSP_9(tin)$, we start again from a matrix grammar in the strong binary normal form, $G = (N, T, S, M, F)$, with k matrices of the form $m_i : (X \to Y, A \to u)$ and h_j matrices of the form $m_i : (X \to Y, B^{(j)} \to \#), j = 1, 2$. In all terminal matrices $(X \to \lambda, A \to u)$ we replace the first rule with $X \to f$, where f is a new symbol.

We construct the P system of degree 9

$$
\begin{array}{rcl}
\Pi & = & (V, T, \mu, (X_0A_0, 1), R_1, \ldots, R_9), \text{ where} \\
V & = & \{X, X', X'', X''' \mid X \in N_1 \cup \{f\}\} \cup T \\
 & \cup & \{X_{i,j}, X'_{i,j}, X''_{i,j}, X'''_{i,j}, \bar{X}_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
 & \cup & \{A_{i,j}, A'_{i,j}, A''_{i,j}, A'''_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
 & \cup & \{X_i^{(j)} \mid m_i : (X \to Y, B^{(j)} \to \#), 1 \le i \le k_j, j = 1, 2\}, \\
\mu & = & [_1[_2[_3\,]_3[_6[_7\,]_7]_6[_8[_9\,]_9]_8]_2[_4[_5\,]_5]_4]_1,
\end{array}
$$

and with the sets of rules given in Table 2 (in all cases, $1 \le j < i$ and $s = 1, 2$):

We also add the rule $f'' \to \lambda(out)$ to set R_1. (In all rules associated with a matrix $m_i : (X \to Y, A \to u)$ where $Y = f$, the corresponding variants of Y are variants of f – primed, barred, etc.)

The work of the system Π is similar to the work of the system from the proof of Lemma 4.1: membranes 6, 7 simulate matrices $m_i : (X \to Y, B^{(1)} \to \#)$, membranes 8, 9 simulate matrices $m_i : (X \to Y, B^{(2)} \to \#)$; the interplay of regions 1, 2, 3, 4, 5 ensures the correct simulation of matrices without appearance checking rules (with the same technique of double subscripts, which ensures the simulation of both rules

Table 2: The rules of the system from the proof of Lemma 4.2

	$m_i : (X \to Y, A \to u)$	$m_i : (X \to Y, B^{(s)} \to \#)$
R_1	$A_{i,j} \to A'_{i,j+1}$ $X_{i,j} \to X'_{i,j+1}$ $A_{i,i} \to A_i$ $X_{i,i} \to \bar{Y}$ $Y'' \to Y$	$Y^{(i)} \to Y^{(i)}$
R_2	$X \to X_{i,0}$ $X'_{i,j} \to X''_{i,j}$ $X'''_{i,j} \to X_{i,j}(out)$ $A' \to A_{i,0}(out)$ $A'_{i,j} \to A_{i,j}$ $A_i \to A_i$ $\bar{Y} \to \bar{Y}$	$X \to Y^{(s)}$
R_3	$X''_{i,j} \to X'''_{i,j}(out)$ $A \to A'(out)$	
R_4	$X'_{i,j} \to X'_{i,j}$ $A'_{i,j} \to A'_{i,j}$ $\bar{Y} \to Y$ $A_i \to u$ $Y' \to Y''(out)$	
R_5	$Y \to Y'(out)$	
R_{4+2s}	$X_{i,0} \to X_{i,0}$ $A_{i,j} \to A_{i,j}$ $X''_{i,j} \to X''_{i,j}$	$Y^{(s)} \to Y$ $Y^{(3-s)} \to Y^{(3-s)}$ $B^{(s)} \to B^{(s)}$ $Y' \to Y(out)$
R_{5+2s}		$Y \to Y'(out)$

of the matrix). The task of checking the details is left to the reader and we only mention that $L(G) = L(\Pi)$.

For the non-extended case, we consider two additional membranes, with labels c_1, c_2, with the following rules:

$$\begin{aligned} R_{c_1} &= \{A \to A \mid X \in N_2\} \cup \{f \to f'''(out)\}, \\ R_{c_2} &= \{f' \to f(out)\}, \end{aligned}$$

and the rule $f'' \to \lambda(out)$ from R_1 is replaced with $f''' \to \lambda(out)$.

All the used symbols are introduced in the alphabet of Π; the terminal alphabet is equal to the total alphabet. A string can exit the system only after passing through membranes c_1, c_2, which are used to check whether or not the string is terminal – only in the affirmative case it can be sent out of the system. □

We close this section with the remarks that the P systems of type *tgo* are particular cases of graph controlled CD grammar systems: after each maximal rewriting in

a membrane, the string leave that region, going up or down in the graph which described the membrane structure; that is, we have a tree controlled CD grammar system. Because we can generate each ET0L language by such a system with three components (using extended systems of type *tgo*), all families obtained in this case are known: CF for one or two components, $ET0L$ for at least three components.

5 In Between PC and P: PC Grammar Systems with Multisets of Strings

We pass now to the bridge between PC grammar systems and P systems. The devices we define can be considered at the same time as tissue P systems with string-objects and communication on request, or as PC grammar systems with multisets of strings present in each component. We use below the second terminology, but the way of presenting our devices is influenced also by the style used in membrane computing.

A PC grammar system (of degree $m \geq 1$) with multisets of strings (in short, an MPC grammar system) is a construct

$$\Pi = (N, K, T, M_1, \ldots, M_n, R_1, \ldots, R_m, i_o),$$

where:

1. N, K, T are pairwise disjoint alphabets, with $K = \{Q_1, \ldots, Q_m\}$ (the elements of K are called *query symbols* and they are associated with the n components of Γ); we denote $V = N \cup T \cup T$;
2. $M_1, \ldots, M_m$ are finite multisets of strings over $N \cup T$;
3. $R_1, \ldots, R_m$ are finite sets of context-free rules of the form $A \to u$, with $A \in N$ and $u \in V^*$;
4. $i_o \in \{1, 2, \ldots, m\}$ (the *master/output* component of Γ).

The work of such a system is a combination of rewriting in a PC grammar system (in the non-returning mode) and of evolution in a tissue P system with string-objects. Specifically, we start from the initial configuration $(M_1, \ldots, M_m)$, and we pass from a configuration $(M'_1, \ldots, M'_m)$, consisting of multisets of strings over $N \cup T$ placed in the m components of the system, to another configuration $(M''_1, \ldots, M''_m)$ in the following way. Each string from each multiset M'_i which can be rewritten according to the rules from $R_i, 1 \leq i \leq m$, is rewritten. This means the use of one rule from R_i, non-deterministically chosen, for each string (at the level of each string, the rewriting is sequential). The strings which cannot be rewritten (no rule can be applied to them) remain unchanged. If no query symbol is introduced (by a given choice of rules), then the resulting multisets of strings are $M''_i, 1 \leq i \leq m$.

Note that the rewriting of strings is maximally parallel, in the sense that all strings which can be rewritten must be rewritten, and that the process is non-deterministic, the choice of rules and the places where the rules are applied can lead to several possible new multisets of strings.

If any query symbol is introduced, then a communication is performed: each symbol Q_j introduced in a string from component i is immediately replaced by all strings from the component j which do not contain query symbols. If in component j there are several strings without query symbols, then each of them is used, hence the string from component i is replicated (with the occurrence of Q_j replaced with strings from component j). If there are several query symbols in the same string from component i, then all of them are replaced (we also say that they are *satisfied*) at the same time, in all possible combinations.

If a query symbol Q_j cannot be satisfied (either component j contains no string, or all strings from component j contain query symbols), then the string containing Q_j is removed (it is like replacing it with the strings from an empty language).

In this way, in each step all query symbols introduced by the rewriting rules disappear, they are either satisfied (replaced by strings without query symbols), or they disappear together with the string which contain them (in the case when they cannot be satisfied). The multisets obtained in this way are $M''_1, \ldots, M''_m$, constituting the next configuration of the system.

Note the difference from the way the communication is defined in a standard PC grammar system and in an MPC grammar system. In our case, a query symbol cannot wait unsatisfied until the requested component contains a string without query symbols (hence the configurations are m-tuples of strings over $N \cup T$, without occurrences of query symbols). This detail, rather natural for the case when we work with multisets of strings, will be crucial for the work of the system from Theorem 6.1, because it entails a nice way of "protecting" a string against communication: if we introduce a query symbol in a string x from a component j, asking for any string z which we know to be over $N \cup T$, even if that string x is requested by a component i, it cannot be moved from component j to component i; the query symbol from component i is either satisfied by other strings from component j, or the string containing it disappears.

We also note that the way the system works corresponds to the non-returning mode from standard PC grammar systems, that is, we do not return to M_j after communicating strings from component j to other components – not even in the case when the component j will contain the empty multiset – but we continue from the remaining strings, if any.

We leave the task of formally defining a transition between two configurations in the system Π to the reader. All terminal strings produced in component i_o during any possible computations in Π is accepted in the language $L(\Pi)$ generated by the system Π. The family of all languages generated in this way by MPC grammar systems of degrees at most $m \geq 1$ is denoted by MPC_mCF; if we use systems of an arbitrary degree, then we replace the subscript m with $*$. (As for standard PC grammar systems, we can consider MPC grammar systems with rules which can be regular, linear, metalinear, etc, that is why we have preserved the indication CF in the notation; the investigation of such classes of systems remains as a research topic.)

We close this section with an **example**, proving the somewhat surprising result that MPC systems with only one component can generate non-context-free languages.

Let us consider the MPC system (of degree 1)

$$\begin{aligned} \Pi &= (\{S, A\}, \{Q_1\}, \{a, b\}, \{(S, 1), (A, 1), (b, 1)\}, R_1, 1), \text{ with} \\ R_1 &= \{S \to aSa,\ S \to Q_1,\ A \to A,\ A \to Q_1Q_1\}. \end{aligned}$$

We start with three strings in the system, S, A, and b. From S we can generate a^nSa^n, for any $n \geq 0$, then we have to replace S with Q_1. If at the same time we have produced Q_1Q_1 from A, then we simultaneously get a^nba^n and bb, both of them included in $L(\Pi)$, and the computation stops.

If A is still present, then we get the strings a^nAa^n and a^nba^n, and A and b are "consumed". The string a^nAa^n will eventually lead to $a^nQ_1Q_1a^n$, and hence to $a^na^nba^na^nba^na^n$, which is in $L(\Pi)$.

If from A we produce Q_1Q_1 while both a^nSa^n and b are present, then we obtain four strings: $a^nSa^na^nSa^n, a^nSa^nb, ba^nSa^n, bb$ (and b is no longer present). The first three strings will be rewritten by the rule $S \to aSa$ for a number of steps, and eventually the rule $S \to Q_1$ is used. The query symbol will be either replaced by a string which contains again S, hence the process continues, or by bb. Consequently, if any terminal string is obtained on this path, then it contains the substring bb.

Consequently, $L(\Pi) \cap a^+ba^+ba^+ = \{a^{2n}ba^{2n}ba^{2n} \mid n \geq 1\}$, which is not a context-free language.

It is highly expected that MPC systems with context-free rules can characterize the recursively enumerable languages, but at this moment we do not have a proof of this assertion.

6 The Efficiency of MPC Grammar Systems

We show now how the possibility of MPC systems to create exponentially many strings in a linear time can be used for solving computationally hard problems in a polynomial time. The framework is that customary in membrane computing (see [11] and, especially, the formal approaches from [12], [13]). Briefly speaking, we work with *confluent* systems, constructed in a *semi-uniform* manner; starting from a given instance of a decision problem, we construct in a polynomial time (by a Turing machine) an MPC system which always stops in a known number of steps, in spite of a possible non-deterministic behavior, and all computations give the same result, which is the answer to the problem (that is, the system is also *sound* and *complete*). The result will be obtained in the output component; for decision problems, the answer will be *yes* if and only if in a specified step this component will contain a specified string.

We illustrate this strategy by solving the satisfiability problems for propositional formulas in the conjunctive normal form, `SAT`.

Theorem 6.1 `SAT` *can be solved in linear time by MPC grammar systems.*

Proof. Let us consider a propositional formula $\gamma = C_1 \wedge \ldots \wedge C_m$, consisting of m clauses $C_j = y_{j,1} \vee \ldots \vee y_{j,k_j}$, $1 \leq j \leq m$, where $y_{j,i} \in \{x_l, \neg x_l \mid 1 \leq l \leq n\}$, $1 \leq i \leq k_j$ (there are used n variables).

For each i, $1 \leq i \leq n$, let us denote by $true(\gamma, t_i)$ the string $c_{j_1} c_{j_2} \ldots c_{j_s}$ indicating the clauses in γ which contain x_i, and by $true(\gamma, f_i)$ the string $c_{j_1} c_{j_2} \ldots c_{j_s}$ of clauses in γ which contain $\neg x_i$ (therefore, these strings are over the alphabet $\{c_i \mid 1 \leq i \leq n\}$).

We construct the MPC system (of degree $3n + 3m + 2$, with the components labelled with $0, 1.1, 1.2, 1.3, \ldots, i.1, i.2, i.3, \ldots, n.1, n.2, n.3, 1, 1', 1'', \ldots, m, m', m'', m+1$)

$$
\begin{array}{rcl}
\Pi & = & (N, K, T, M_0, M_{1.1}, \ldots, M_{m''}, M_{m+1}, R_0, R_{1.1}, \ldots, R_{m''}, R_{m+1}, m+1), \\
N & = & \{a_i \mid 1 \leq i \leq 3n + 2m + 1\} \\
 & \cup & \{a'_i,\ a''_i \mid 1 \leq i \leq 2n\} \\
 & \cup & \{t_i,\ f_i \mid 1 \leq i \leq n\} \\
 & \cup & \{c_i \mid 1 \leq i \leq m\} \\
 & \cup & \{b\}, \\
K & = & \{Q_0,\ Q_{m+1}\} \\
 & \cup & \{Q_{i.1},\ Q_{i.2},\ Q_{i.3} \mid 1 \leq i \leq n\} \\
 & \cup & \{Q_i,\ Q_{i'},\ Q_{i''} \mid 1 \leq i \leq m\}, \\
T & = & \{d\}, \\
M_0 & = & \{b\}, \\
M_{i.1} & = & M_{i.2} = M_j = M_{j'} = M_{m+1} = \{a_1\},\ \text{for all } 1 \leq i \leq n, 1 \leq j \leq m, \\
M_{i.3} & = & \{a'_i,\ a''_i\},\ \text{for all } 1 \leq i \leq n, \\
M_{i''} & = & \{d\},\ \text{for all } 1 \leq i \leq m, \\
R_0 & = & \emptyset, \\
R_{1.1} & = & \{a_1 \to Q_0,\ b \to t_1 b\}, \\
R_{1.2} & = & \{a_1 \to Q_0,\ b \to f_1 b\}, \\
R_{1.3} & = & \{a'_1 \to a'_2,\ a''_1 \to a''_2,\ a'_2 \to Q_{1.1},\ a''_2 \to Q_{1.2}\}, \\
R_{i.1} & = & \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(i-1)\} \\
 & \cup & \{a_{2i-1} \to Q_{i-1.3},\ b \to t_i b\},\ \text{for all } i = 2, 3, \ldots, n-1, \\
R_{i.2} & = & \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(i-1)\} \\
 & \cup & \{a_{2i-1} \to Q_{i-1.3},\ b \to f_i b\},\ \text{for all } i = 2, 3, \ldots, n-1, \\
R_{i.3} & = & \{a'_j \to a'_{j+1},\ a''_j \to a''_{j+1} \mid 1 \leq j \leq 2i-1\} \\
 & \cup & \{a'_{2i} \to Q_{i.1},\ a''_{2i} \to Q_{i.2}\},\ \text{for all } i = 2, 3, \ldots, n-1, \\
R_{n.1} & = & \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(n-1)\} \\
 & \cup & \{a_{2n-1} \to Q_{n-1.3},\ b \to t_n\}, \\
R_{n.2} & = & \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(n-1)\} \\
 & \cup & \{a_{2n-1} \to Q_{n-1.3},\ b \to f_n\}, \\
R_{n.3} & = & \{a'_j \to a'_{j+1},\ a''_j \to a''_{j+1} \mid 1 \leq j \leq 2n-1\} \\
 & \cup & \{a'_{2n} \to Q_{n.1},\ a''_{2n} \to Q_{n.2}\} \\
 & \cup & \{t_i \to true(\gamma, t_i),\ f_i \to true(\gamma, f_i) \mid 1 \leq i \leq n\}, \\
R_1 & = & \{a_j \to a_{j+1} \mid 1 \leq j \leq 3n\}
\end{array}
$$

$$
\begin{aligned}
& \cup \ \{a_{3n+1} \to Q_{n.3},\ c_1 \to Q_{1''}\}, \\
R_{1'} &= \{a_j \to a_{j+1} \mid 1 \le j \le 3n+1\} \\
& \cup \ \{a_{3n+2} \to Q_1\}, \\
R_i &= \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2(i-1)\} \\
& \cup \ \{a_{3n+2(i-1)+1} \to Q_{i-1},\ c_i \to Q_{i''}\}, \text{ for all } i = 2,3,\ldots,m, \\
R_{i'} &= \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2(i-1)+1\} \\
& \cup \ \{a_{3n+2(i-1)+2} \to Q_i\}, \text{ for all } i = 2,3,\ldots,m, \\
R_{i''} &= \emptyset, \text{ for all } i = 1,2,\ldots,m, \\
R_{m+1} &= \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2m\} \\
& \cup \ \{a_{3n+2m+1} \to Q_m\}.
\end{aligned}
$$

The system works as follows. All symbols a_i, a_i', a_i'' are counters, used for the synchronization of the computation. The components $i.1, i.2, i.3$ are used for expanding the variable x_i, so that in component $n.3$ we get all 2^n truth-assignments for the n variables in the form of strings $\alpha_1\alpha_2\ldots\alpha_n$, with $\alpha_i \in \{t_i, f_i\}$, with t_i indicating the value *true* and f_i indicating the value *false*. In the same component $n.3$ one also identifies for each truth-assignment the sequence of clauses which are satisfied; this leads to 2^n strings over the alphabet $\{c_i \mid 1 \le i \le m\}$. These strings are examined in components i, i', i'' in order to see whether at least one string exists which contains all $c_i, 1 \le i \le m$. If such a string exists, then a string over $\{c_i \mid 1 \le i \le m\} \cup \{d\}$ will be present in component $m+1$ after step $3n+2m+1$, indicating that the formula is satisfiable; if no string of this form will be present in component $m+1$ after step $3n+2m+1$, then γ is not satisfiable.

Let us now examine in a closer manner the computations in Π.

In the first step, both components 1.1, 1.2 introduce Q_0, hence b from component 0 is replicated and sent to the two components; here, b is replaced by t_1b and f_1b, respectively, corresponding to the *true* and *false* values for x_1. Note that b is reproduced. In the first step, the component 1.3 increases the counters, while in the second step one uses the rules $a_2' \to Q_{1.1}, a_2'' \to Q_{1.2}$. The strings from components 1.1 and 1.2 are moved to component 1.3. From now on, components 1.1 and 1.2 will be empty.

During the first two steps, components 2.1 and 2.2 (actually, all components of the system where counters are present) just increase the counters. In step 3, both components 2.1 and 2.2 ask for the strings of component 1.3, hence these strings are replicated and sent to components 2.1 and 2.2. We have in each of these components the strings t_1b, f_1b, hence in the fourth step we use the rule $b \to t_2b$ in component 2.1 and the rule $b \to f_2b$ in component 2.2. The strings we obtain are t_1t_2b, f_1t_2b in component 2.1, and t_1f_2b, f_1f_2b in component 2.2. At the same time (hence in step 4), component 2.3 asks for all these strings, hence component 2.2 will contain now the strings $t_1t_2b, t_1f_2b, f_1t_2b, f_1f_2b$, which means that the first two variables were expanded (in four steps).

In general, in $2i$ steps, we expand the first i variables $x_1, \ldots, x_i$, and this is true also for $i = n$. After obtaining all 2^n truth-assignments $\alpha_1\alpha_2\ldots\alpha_n$, with $\alpha_i \in \{t_i, f_i\}$, in component $n.3$ we use the rules $t_i \to true(\gamma, t_i)$, $f_i \to true(\gamma, f_i)$.

Each string has length n, hence we will have n steps to perform in order to replace each t_i, f_i by the sequence of satisfied clauses. (Note that it is possible that the same t_i, f_i satisfies none, one, or several clauses, hence the strings over $\{c_j \mid 1 \leq j \leq m\}$ we obtain now can be of any length between zero – when a truth-assignment satisfies no clause – to nm – the case of a truth-assignment where each t_i, f_i satisfies all clauses.)

After these $3n$ steps, we pass to examining all the obtained strings, checking whether at least one exists which contains all $c_i, 1 \leq i \leq m$. During the $3n$ steps, components j, j', $j = 1, 2, \ldots, m$, have just increased the counters. In step $3n+1$, in component 1 we use the rule $a_{3n+1} \to Q_{n.3}$, which brings all strings from component $n.3$ into component 1. In all first $3n+1$ steps, component $1'$ has increased the counters. In step $3n+2$ we can use the rule $c_1 \to Q_{1''}$ in component 1 (and this is obligatory for each string which contains the symbol c_1) and, simultaneously, the rule $a_{3n+2} \to Q_1$ in component $1'$. In this way, all strings which contain at least one occurrence of c_1 will contain now the query symbol $Q_{1''}$, while the strings which do not contain c_1 remain unchanged (hence they contain no query symbol). Therefore, all strings which do not contain the query symbol are "cleaned" from component 1, and moved to component $1'$. Simultaneously, all strings which were "protected" by the symbol $Q_{1''}$ will replace $Q_{1''}$ by d and will remain in component 1.

In the next step, component 2 will request all strings from component 1 (in the first $3n+2$ steps, this component has only increased the counters – like all components i, i' for $i = 2, 3, \ldots, m, m+1$). The process is iterated, checking in the same manner whether $c_2, c_3, \ldots, c_m$ are present. The checking takes two steps for each clause. After $2m$ steps (added to the $3n$ steps of producing the strings of satisfied clauses) we have in component m the strings which correspond to the truth-assignments which have satisfied all clauses – maybe none, if no such a truth-assignment exists. Therefore, if any string survives the checking phase, then in step $3n+2m+1$, when component $m+1$ uses the rule $a_{3n+2m+1} \to Q_m$, we can see in component $m+1$ whether the formula is satisfiable: a string arrives here if and only if the formula is satisfiable.

We close this discussion by pointing out that the conditions we have stated at the beginning of this section in order to claim that we have solved the problem are fulfilled: the system can be constructed in a polynomial time (it has a polynomial size, in terms of the number of used symbols, number of rules, total length of rules), it is confluent (some non-determinism is allowed in the place of using the rules $t_i \to true(\gamma, t_i)$, $f_i \to true(\gamma, f_i)$, and then in the place of using the rules $c_i \to Q_{i''}, 1 \leq i \leq m$, but the result is the same in all cases, because there are n steps of using the first type of rules, rewriting strings of length n, and m steps of using the second type of rules, which is enough for checking the presence of all clauses, hence each computation gives the same result), sound and complete (the problem has a solution if and only if the system indicates that a solution exists). With the observation that the result is obtained in a linear time, the proof is complete. □

Let us note that the system constructed in the previous proof depends only on n and m, with one exception, the rules $t_i \to true(\gamma, t_i)$, $f_i \to true(\gamma, f_i)$, for $1 \leq i \leq n$, from $R_{n.3}$, which directly depend on the instance of `SAT` we are handling. Therefore, if we consider these rules as an input to the system, then the construction can be

considered uniform; if this convention is not allowed, then the instance must be codified by means of strings which have to be introduced in the initial configuration of the system in a specified component (added to the multiset of that component). The construction of such a system remains as an open problem.

7 Further Remarks, Further Research Topics

As we have stressed several times in the paper, this is only a first approach towards a systematic investigation of the possibilities to bridge the grammar systems area and the membrane computing area. The benefit will be mutual, and this was already proved by the previous results.

Several open problems and research topics were already mentioned in the paper, but several others remain to be considered. We only mention a few topics which seems to be both natural and of interest: Consider classes of MPC grammar systems like in the case of usual PC grammar systems, that is, centralized or non-centralized, returning or non-returning. We have considered here strings; what about working with multisets of symbol-objects (hence having a computing device closer to tissue P systems, namely, tissue P systems with communication by request)?

And, of course, after investigating the possible relationships between CD or PC grammar systems and P systems, we can also consider other classes of grammar systems, such as the colonies [8], eco-grammar systems [2] (problem Q36 from [11] already asks this), networks of language processors [4]. The case of eco-grammar systems would mean having evolution rules not only in the components/cells, but also in the environment, a case not addressed yet in membrane computing, although it looks rather realistic. In turn, from networks of language processors we can borrow the idea of filters for the communicated strings (or symbols, when working with multisets of symbols). This can be of a special interest in the case of multisets of symbols, because we can request from a component only part of the available symbols.

Acknowledgements. Work done during a stay of the author as a visiting scientist at MTA SZTAKI EU Centre of Excellence, Budapest. Part of the results reported here were obtained in collaboration with E. Csuhaj-Varjú and Gy. Vaszil and they will constitute the starting point of more elaborated papers, now in preparation, about relationships of CD and PC grammar systems and P systems. It is also worth mentioning a forthcoming paper by E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, and Gy. Vaszil, about P colonies.

References

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

[2] E. Csuhaj-Varjú, A. Kelemenová, J. Kelemen, Gh. Păun, Eco-grammar systems – A grammatical framework for life-like interactions, *Artificial Life*, 3 (1997), 1–28.

[3] E. Csuhaj-Varjú, Gh. Păun, Gy. Vaszil, PC grammar systems with five components can generate all recursively enumerable languages, *Theoretical Computer Sci.*, 299, 1-3 (2003), 785–794.

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

[5] Z. Dang, O.H. Ibarra, On P systems operating in sequential and limited parallel modes, *DCFS 2004*.

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

[7] O.H. Ibarra, H.-C. Yen, Z. Dang, The power of maximal parallelism in P systems, *DLT 2004*.

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

[9] N. Mandache, On the computational power of context-free PC grammar systems, *Theoretical Computer Science*, 237 (2000), 135–148.

[10] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).

[11] Gh. Păun, *Computing with Membranes: An Introduction*, Springer-Verlag, Berlin, 2002.

[12] M. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas*, Editorial Kronos, Sevilla, 2002.

[13] M. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity classes in cellular computation with membranes, *Natural Computing*, 2, 3 (2003), 265–285.

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

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

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 276 - 286, 2004.

Image Splicing Grammar Systems

A. Roslin Sagaya Mary

Research Group on Mathematical Linguistics, Rovira i Virgili University,
Pl. Imperial Tarraco 1, 43005 Tarragona, Spain
writetoroslin@yahoo.co.uk

K. G. Subramanian

Department of Mathematics, Madras Christian College
Tambaram, Chennai 600 059, India
kgsmani1948@yahoo.com

K.S.Dersanambika

Department of Computer Science and Engineering,
Indian Institute of Technology,Madras
Chennai 600 036,India
dersanapdf@yahoo.com

Abstract

Motivated by the study of Dassow and Mitrana [3], we consider Grammar Systems that describe Images or Pictures of rectangular arrays with the components consisting of two-dimensional Grammars and domino splicing rules [8] with the grammars working in parallel and splicing rules acting on arrays of two components, yielding rectangular arrays of symbols. The resulting systems are called Image Splicing Grammar Systems (ISGS). Certain Properties of ISGS with different component grammars are obtained.

1 Introduction

The theory of Grammar systems is a well-investigated field of formal language theory, providing a theoretical framework for modelling various kinds of multi-agent systems at the symbolic level [2]. A grammar system consists of several grammars or other language identifying mechanisms, that cooperate according to some well-defined protocol. The components of the system correspond to the agents, the current string(s) in generation to a symbolic environment, and the system's behaviour is represented by the language. Among a variety of grammar system models, Parallel Communicating Grammar Systems, in which the components are generative grammars working on their own sentential forms in parallel and communicating with each other by sending their sentential forms by request, have been of intensive study [1, 2].

On the other hand, there has been a lot of interest in the study of formal language theory applied to DNA computing. Head [6] defined splicing systems motivated

by the behaviour of DNA sequences under the action of restriction enzymes and ligases. The splicing systems make use of a new operation, called splicing on strings of symbols. Theoretical investigation of splicing on strings has been extensively done by different researchers [7].

A new type of Parallel Communicating grammar systems has been introduced in [3] by replacing communication by splicing of strings.

In syntactic approaches to generation and recognition of image or picture patterns, considered as digitized arrays, several two-dimensional grammars have been proposed and studied [9]. As a simple and effective extension of the operation of splicing on strings a new method of splicing on images of rectangular arrays is introduced in [8] . The idea here is that two rectangular arrays are column spliced or row spliced by using the domino splicing rules in parallel.

Freund [4] has introduced and investigated cooperating distributed array grammar systems extending the concept of cooperation in string grammar systems and using array grammars. Here, motivated by the study of Dassow and Mitrana [3], we consider Grammar Systems that describe Images or Pictures of rectangular arrays. The components of the Grammar system consist of two-dimensional Grammars [5, 10, 9] and domino splicing rules [8] with the grammars working in parallel and splicing rules acting on arrays of two components yielding rectangular arrays of symbols. The resulting systems are called Image Splicing Grammar Systems. Different component grammars such as Regular Matrix grammars [5, 10], Context-free Matrix grammars [5, 10], are considered and properties such as generative power, comparison etc. are obtained.

2 Preliminaries

The basic notions and notations on arrays are now recalled [5, 10].

Let Σ be a finite alphabet. Σ^* is the set of all words over Σ including the empty word λ. An image or a picture A over Σ is a rectangular $m \times n$ array of elements of Σ of the form

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

or in short $A = [a_{ij}]_{m\times n}$. We write an array A without enclosing it in square brackets when there is no confusion. The set of all images is denoted by Σ^{**}. A picture language or a two dimensional language over Σ is a subset of Σ^{**}.

Definition 1 For an array A of dimension $m \times n$ and an array B of dimension $m' \times n'$, the column catenation $A\Phi B$ is defined only when $m = m'$ and the row catenation $A\Theta B$ is defined only when $n = n'$.

Informally speaking, in row catenation $A\Theta B$, B is attached below A. In column catenation $A\Phi B$, B is attached to the right of X. We refer to [5, 10]for a formal definition of column and row catenations of rectangular arrays.

Definition 2 (Two-Dimensional Matrix Grammars) A $2D$ **matrix grammar** is a $2-$tuple (G_1, G_2) where
$G_1 = (H_1, I_1, P_1, S)$ is a Regular, CF or CS grammar,
H_1 is a finite set of horizontal nonterminals,
$I_1 = \{S_1, S_2, \cdots, S_k\}$, a finite set of intermediates, $H_1 \cap I_1 = \emptyset$,
P_1 is a finite set of production rules called horizontal production rules,
S is the start symbol, $S \in H_1$,
$G_2 = (G_{21}, G_{22}, \cdots, G_{2k})$ where
$G_{2i} = (V_{2i}, T, P_{2i}, S_i), 1 \leq i \leq k$ are regular grammars,
V_{2i} is a finite set of vertical nonterminals, $V_{2i} \cap V_{2j} = \emptyset,\ i \neq j$,
T is a finite set of terminals,
P_{2i} is a finite set of right linear production rules of the form $X \longrightarrow aY$ or $X \longrightarrow a$ where $X, Y \in V_{2i}, a \in T$
$S_i \in V_{2i}$ is the start symbol of G_{2i}.

The type of G_1 gives the type of G , so we speak about regular, context-free, context sensitive $2D$ matrix grammars if G_1 is regular, context-free, context sensitive respectively. Derivations are defined as follows: First a string $S_{i1}S_{i2} \cdots S_{in} \in I_1^*$ is generated horizontally using the horizontal production rules of P_1 in G_1. That is, $S \Rightarrow S_{i1}S_{i2} \cdots S_{in} \in I_1^*$. Vertical derivations proceed as follows: We write

$$\begin{array}{c} A_{i1} \cdots A_{in} \\ \Downarrow \\ a_{i1} \cdots a_{in} \\ B_{i1} \cdots B_{in} \end{array}$$

if $A_{ij} \rightarrow a_{ij}B_{ij}$ are rules in $P_{2j}, 1 \leq j \leq n$. The derivation terminates if $A_j \rightarrow a_{mj}$ are all terminal rules in G_2.

The set $L(G)$ of all matrices generated by G consists of all $m \times n$ arrays $[a_{ij}]$ such that $1 \leq i \leq m,\ 1 \leq j \leq n$ and $S \Rightarrow^*_{G_1} S_{i1}S_{i2} \cdots S_{in} \Rightarrow^*_{G_2} [a_{ij}]$. We denote the picture language classes of regular, CF, CS 2DMatrix grammars by 2DRML, 2DCFML, 2DCSML respectively.

We next recall the notion of splicing on strings [6, 7].

Definition 3 Let V be an alphabet and #, \$ two special symbols not in V. A splicing rule over V is a string of the form $r = u_1\#u_2\$u_3\#u_4$, where $u_i \in V^*, 1 \leq i \leq 4$. For such a rule r and strings $x, y, z \in V^*$, we write

$(x, y) \vdash_r z$ iff $x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ z = x_1u_1u_4y_2$
for some $x_1, x_2, y_1, y_2 \in V^*$. We say that z is obtained by splicing x, y, as indicated by the rule r; u_1u_2 and u_3u_4 are called the sites of the splicing.

We now recall the notions of domino splicing rules and Splicing of arrays using these rules [8].

Definition 4 Let V be an alphabet. $\#, \$$ are two special symbols, not in V. A domino over V is of the form $\boxed{\begin{array}{c} a \\ \hline b \end{array}}$ or $\boxed{a \mid b}$ for some $a, b \in V$

A domino column splicing rule over V is of the form $r = \alpha_1 \# \alpha_2 \$ \alpha_3 \# \alpha_4$ where each $\alpha_i = \boxed{\begin{array}{c} a \\ \hline b \end{array}}$ for some $a, b \in V \cup \{\#\}$.

A domino row splicing rule over V is of the form $r = \beta_1 \# \beta_2 \$ \beta_3 \# \beta_4$ where each $\beta_i = \boxed{a \mid b}$ for some $a, b \in V \cup \{\#\}$.

We refer to $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of a column splicing rule $r = \alpha_1 \# \alpha_2 \$ \alpha_3 \# \alpha_4$ as the first, second, third and fourth dominoes of r respectively. Similarly for a row splicing rule $r = \beta_1 \# \beta_2 \$ \beta_3 \# \beta_4$. $\beta_1, \beta_2, \beta_3, \beta_4$ are the first, second, third and fourth dominoes of r respectively.

Given two arrays X and Y of sizes $m \times p$ and $m \times q$ respectively,

$$X = \begin{array}{cccccc} a_{11} & \cdots & a_{1,j} & a_{1,j+1} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2,j} & a_{2,j+1} & \cdots & a_{2p} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m,j} & a_{m,j+1} & \cdots & a_{mp} \end{array} ,$$

$$Y = \begin{array}{cccccc} b_{11} & \cdots & b_{1,k} & b_{1,k+1} & \cdots & b_{1q} \\ b_{21} & \cdots & b_{2,k} & b_{2,k+1} & \cdots & b_{2q} \\ \vdots & \ddots & \cdots & \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{m,k} & b_{m,k+1} & \cdots & b_{mq} \end{array}$$

a_{ir}, $b_{is} \in V$, for $1 \leq i \leq m$, $1 \leq r \leq p$, $1 \leq s \leq q$. We write $(X, Y) \quad |\overset{\Phi}{-} \quad Z$ if there is a sequence of column splicing rules $r_1, r_2, ...r_m$ (not necessarily all different) such that

$$r_i = \boxed{\begin{array}{c} a_{i,j} \\ \hline a_{i+1,j} \end{array}} \quad \# \quad \boxed{\begin{array}{c} a_{i,j+1} \\ \hline a_{i+1,j+1} \end{array}} \quad \$ \quad \boxed{\begin{array}{c} b_{i,k} \\ \hline b_{i+1,k} \end{array}} \quad \# \quad \boxed{\begin{array}{c} b_{i,k+1} \\ \hline b_{i+1,k+1} \end{array}}$$

for all i, $1 \leq i \leq m-1$ and for some j, k $1 \leq j \leq p-1$, $1 \leq k \leq q-1$ and

$$Z = \begin{array}{cccccc} a_{11} & \cdots & a_{1,j} & b_{1,k+1} & \cdots & b_{1q} \\ a_{21} & \cdots & a_{2,j} & b_{2,k+1} & \cdots & b_{2q} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m,j} & b_{m,k+1} & \cdots & b_{mq} \end{array}$$

In other words, we can imagine that a 2×1 window is moved down the jth column of X The sequence of dominoes collected are the first dominoes of the rules $r_1, r_2, ..., r_m$ (not all necessarily different). When a 2×1 window is moved down the $j + 1^{st}$ column of X the sequence of dominoes collected are the second dominoes of the rules $r_1, r_2, ..., r_m$. Likewise for the kth and $k + 1^{st}$ columns of Y When such rules exist in the system, the column splicing of the arrays X and Y amounts to the array X being vertically "cut" between j^{th} and $j + 1^{st}$ columns and the array Y between k^{th} and $k + 1^{st}$ columns and the resulting left subarray of X "pasted" (column

catenated) with the right subarray of Y to yield Z We now say that Z is obtained from X and Y by domino column splicing in parallel.

We can similarly define row splicing operation of two arrays U and V of sizes $p \times n$ and $q \times n$, using row splicing rules to yield an array W.

$$U = \begin{matrix} c_{11} & c_{12} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r,1} & c_{r,2} & \cdots & c_{r,n} \\ c_{r+1,1} & c_{r+1,2} & \cdots & c_{r+1,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pn} \end{matrix}, \qquad V = \begin{matrix} d_{11} & d_{12} & \cdots & d_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{s,1} & d_{s,2} & \cdots & d_{s,n} \\ d_{s+1,1} & d_{s+1,2} & \cdots & d_{s+1,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{q1} & d_{q2} & \cdots & d_{qn} \end{matrix}$$

c_{rj}, $d_{sj} \in V$, for $1 \leq j \leq n$, $1 \leq r \leq p$, $1 \leq s \leq q$.

We write $(U, V) \quad |\overset{\Theta}{-} \quad W$ if there is a sequence of row splicing rules $r_1, r_2, \cdots r_n$ (not necessarily all different) such that

$$r_j = \boxed{c_{r,j} \mid c_{r,j+1}} \quad \# \quad \boxed{c_{r+1,j} \mid c_{r+1,j+1}} \quad \$ \quad \boxed{d_{s,j} \mid d_{s,j+1}} \quad \# \quad \boxed{d_{s+1,j} \mid d_{s+1,j+1}}$$

for all j, $1 \leq j \leq n-1$ and for some r, s $1 \leq r \leq p-1$, $1 \leq s \leq q-1$ and

$$W = \begin{matrix} c_{11} & c_{12} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r,1} & c_{r,2} & \cdots & c_{r,n} \\ d_{s+1,1} & d_{s+1,2} & \cdots & d_{s+1,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{q1} & d_{q2} & \cdots & d_{qn} \end{matrix}$$

As done for the column splicing of arrays, we can imagine 1×2 windows being moved over respective rows. The row splicing of the arrays U and V can be thought of as U being horizontally "cut" between the rth and $r+1^{st}$ rows and V between s^{th} and $s+1^{st}$ rows and the upper subarray of U "pasted" (row catenated) to the lower subarray of V to yield W We now say that W is obtained from U and V by domino row splicing in parallel.

We illustrate with an example.

Example 1 Let $V = \{a, b\}$,

$$R_c = \{p_1 : \boxed{\begin{matrix} a \\ b \end{matrix}} \quad \# \quad \$ \quad \# \quad \boxed{\begin{matrix} b \\ a \end{matrix}}$$

$$p_2 : \boxed{\begin{matrix} b \\ a \end{matrix}} \quad \# \quad \$ \quad \# \quad \boxed{\begin{matrix} a \\ b \end{matrix}} \quad \}$$

$$R_r = \{q_1 : \boxed{a \mid b} \quad \# \quad \$ \quad \# \quad \boxed{b \mid a}$$

$$q_2 : \boxed{b \mid a} \quad \# \quad \$ \quad \# \quad \boxed{a \mid b} \quad \}$$

Column splicing in parallel of an array with itself using the rules given is shown below:

$$\begin{array}{cc|} a & b \\ b & a \end{array} \quad \begin{array}{|cc} a & b \\ b & a \end{array} \quad \begin{array}{|c} \Phi \\ \hline \end{array} \quad \begin{array}{cccc} a & b & a & b \\ b & a & b & a \end{array}$$

Likewise, row splicing in parallel of an array with itself using the rules given is shown below:

$$\begin{array}{cccc} a & b & a & b \\ b & a & b & a \\ \hline \end{array} \quad \begin{array}{cccc} a & b & a & b \\ b & a & b & a \\ \hline a & b & a & b \\ b & a & b & a \end{array} \quad \begin{array}{c} \Theta \\ \hline \end{array}$$

A vertical bar '| ' or a horizontal bar '—' indicates the place where splicing is done.

3 Image Splicing Grammar System

We now introduce the notion of Image Splicing grammar system in which the component grammars are 2DMatrix grammars.

Definition 5 An Image Splicing Grammar system is a construct
$\Gamma = (V_h, \Sigma_I, V_v, T, (S_1, R_1^h, R_1^v), ..., (S_n, R_n^h, R_n^v), M)$ where,

V_h is a finite set of variables called horizontal variables;
V_v is a finite set of variables called vertical variables;
$\Sigma_I \subseteq V_v$ is a finite set of intermediates;
T is a finite set of terminals;
$S_i, 1 \leq i \leq n$ is the start symbol of the corresponding horizontal component;
$R_i^h, 1 \leq i \leq n$ is a finite set of rules called horizontal productions
and the rules can be regular or context free or context sensitive;
$R_i^v, 1 \leq i \leq n$ is a finite set of right linear rules called vertical productions;
M is a finite set of domino column or row splicing rules of the form

$$m = \alpha_1 \ \# \ \alpha_2 \ \$ \ \alpha_3 \ \# \ \alpha_4 \text{ or } \beta_1 \ \# \ \beta_2 \ \$ \ \beta_3 \ \# \ \beta_4$$

where $\alpha_i = \begin{array}{|c|} \hline a \\ \hline b \\ \hline \end{array}$ and $\beta_i = \begin{array}{|c|c|} \hline c & d \\ \hline \end{array}$ for some $a, b, c, d \in V_v \cup \{T\} \cup \{\lambda\}$.

The derivations take place in two phases as follows :

Each component grammar generates a word called intermediate word, over intermediates starting from its own start symbol and using its horizontal production rules ; the derivations in this phase are done with the component grammars working in parallel.

In the second phase any of the following steps can take place :

(i) each component grammar can rewrite as in a two dimensional matrix grammar using the vertical rules, starting from its own intermediate word generated in

the first phase. (The component grammars rewrite in parallel).Note that the component grammars together terminate or together continue rewriting in the vertical direction.

(ii) At any instant the array X generated in the ith component for some $1 \leq i \leq n$ and the array Y generated in the jth component for some $1 \leq j \leq n$ can be spliced using column / row domino splicing rules as in definition 4, thus yielding array Z in i^{th} component and W in the jth component; In fact Z will have a prefix of X column concatenated with a suffix of Y and W will have a prefix of Y, column concatenated with a suffix of X, the prefixes and suffixes being given by the splicing rules. In any other components (other than i^{th}, j^{th} components), the arrays generated at this instant will remain unchanged during this splicing process.

There is no priority between steps (i) and (ii).

The language $L_i(\Gamma)$ generated by the ith component of Γ consists of all arrays, generated over T, by the derivations described above.

This language will be called the *individual language* of the system and we may choose this to be the language of the first component and $L_t(\Gamma) = \bigcup_{i=1}^{n} L_i(\Gamma)$ as the *total language*. The family of individual languages generated by ISGS with n components of type X for $X \in \{REG, CF\}$ is denoted by $I_{isgs}L_n(X)$, and the corresponding family of total languages by $T_{isgs}L_n(X)$ respectively and $Y_{isgs}L_n(X)$ when $Y \in \{I, T\}$. We basically deal with individual languages although the results obtained apply to total languages as well.

Example 2 Let $\Gamma = (\{S, X\}, \{A, B, C\}, \{A, B, C, D\}\{., x\}, (S, R^h, R^v), (S, R^h, R^v), (S, R^h, R^v), M)$
where
$R^h = \{S \to AX, X \to BX, X \to C$
$R^v = \{A \to xA, A \to x, B \to xD, D \to .D, D \to x, C \to xC, C \to x\}$.

$$
\begin{array}{l}
M = \{ \begin{array}{|c|}\hline x \\ \hline . \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \\
\qquad \begin{array}{|c|}\hline . \\ \hline . \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \\
\qquad \begin{array}{|c|}\hline . \\ \hline D \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline C \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline A \\ \hline\end{array} \\
\qquad \begin{array}{|c|c|}\hline x & . \\ \hline\end{array} \ \# \ \begin{array}{|c|c|}\hline A & D \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \\
\qquad \begin{array}{|c|c|}\hline . & . \\ \hline\end{array} \ \# \ \begin{array}{|c|c|}\hline D & D \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \\
\qquad \begin{array}{|c|c|}\hline . & x \\ \hline\end{array} \ \# \ \begin{array}{|c|c|}\hline D & D \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|c|}\hline x & x \\ \hline\end{array} \ \}
\end{array}
$$

The horizontal rules in a component generate intermediate words of the form AB^nC with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangle pictures of (.)'s surrounded or bordered

by x's except the bottom border which will be of the form AD^nC. At this stage with domino splicing rules, column or row splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. In fact any picture generated in the individual language of this Image splicing grammar system will be either (i) rectangular pictures in which any row, except the first and the last, will be of the form $(x(.)^n)^k x$ for some $k\epsilon\{1,2,3\}$ or (ii)rectangular pictures in which any column, except the first and the last, will have a similar feature or (iii)simply a column of $x's$. Two such pictures obtained are shown in Figures 1(a) and 1(b).

```
x x x x x x x x x x x x x x x x
x . . . . x . . . . x . . . . x
x . . . . x . . . . x . . . . x
x . . . . x . . . . x . . . . x
x x x x x x x x x x x x x x x x
```

Figure 1(a). A Picture of Example 2

```
x x x x x
x . . . x
x . . . x
x . . . x
x x x x x
x . . . x
x . . . x
x . . . x
x . . . x
x x x x x
x . . . x
x . . . x
x . . . x
x . . . x
x x x x x
```

Figure 1(b). Another Picture of Example 2

Example 3

Let $\Gamma = (\{S, X\}, \{A, B, E\}, \{A, B, C, D, E\}\ \{., x\}, (S, R^h, R^v), (S, R^h, R^v), (S, R^h, R^v), M)$ where

$R^h = \{S \to EXE, X \to AXB, X \to AB\}$
$R^v = \{A \to aC, C \to .C, C \to a,$
$\quad B \to bD, D \to .D, D \to b,$

$E \to xE, E \to x\}$

and $M = \{ \begin{array}{c} b \\ . \end{array} \# \begin{array}{c} x \\ x \end{array} \$ \begin{array}{c} x \\ x \end{array} \# \begin{array}{c} a \\ . \end{array}$

$\begin{array}{c} . \\ . \end{array} \# \begin{array}{c} x \\ x \end{array} \$ \begin{array}{c} x \\ x \end{array} \# \begin{array}{c} . \\ . \end{array}$

$\begin{array}{c} . \\ D \end{array} \# \begin{array}{c} x \\ E \end{array} \$ \begin{array}{c} x \\ E \end{array} \# \begin{array}{c} . \\ C \end{array} \}$

The horizontal rules generate in a component intermediate words of the form EA^nB^nE with the same value of $n \geq 1$ at a time. The vertical rules of the components generate from an intermediate word rectangle pictures of (.)'s bordered on the top by words of the form xa^mb^mx, on the bottom by words of the form EC^mD^mE , the leftmost column being a column of x's ending with E and the rightmost column being a column of x's ending with E. At this stage with domino splicing rules, column splicing of the array in a component with the array in another component can take place before rewriting is terminated in the components with terminating vertical rules. One such picture obtained is shown in Figure 2.

$$\begin{array}{ccccccccccccc}
\mathbf{x} & a & a & b & b & a & a & b & b & a & a & b & b & \mathbf{x} \\
\mathbf{x} & . & . & . & . & . & . & . & . & . & . & . & . & \mathbf{x} \\
\mathbf{x} & . & . & . & . & . & . & . & . & . & . & . & . & \mathbf{x} \\
\mathbf{x} & . & . & . & . & . & . & . & . & . & . & . & . & \mathbf{x} \\
\mathbf{x} & a & a & b & b & a & a & b & b & a & a & b & b & \mathbf{x}
\end{array}$$

Figure 2. A Picture of Example 3.

Theorem 1 For $Y \in \{I, T\}$,

1. $2DRML = Y_{isgs}L_1(REG)$
2. $2DRML \subset Y_{isgs}L_2(REG)$
3. $2DCFML = Y_{isgs}L_1(CF)$
4. $2DCFML \subset Y_{isgs}L_2(CF)$

Statements (1) and (3) are obvious. The proper inclusion in statement (2) is a consequence of Example 2. In fact the pictures in Figures $1(a)$ and $1(b)$ cannot be generated by any $2DRMG$ as both the rules in both the horizontal and vertical phases are only regular rules. Likewise the proper inclusion in statement (4) is a consequence of Example 3 since the rules in the horizontal phase of a $2DCFMG$ are only CF rules and so the pictures as in Figure 2 require CS rules in the first phase.

Example 4 Let $\Gamma = (\{S_1, \cdots S_n, X\}, \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, \}, \{A_1, \cdots, A_n, B_1, \cdots, B_n, C_1, \ldots, C_n, D_1, \ldots, D_n\},$

$\{., x, a, b\}, (S, R^{h_1}, R^{v_1}), (S, R^{h_2}, R^{v_2}), \cdots, (S, R^{h_n}, R^{v_n}), M)$
where

$R^{h_1} = \{S_1 \to A_1X, X \to B_1X, X \to C_1$
$R^{v_1} = \{A_1 \to xA_1, A_1 \to x, B_1 \to aD_1, D_1 \to .D_1, D_1 \to a, C_1 \to xC_1, C_1 \to x, D_i \to a \ if \ i \geq 2 \ and \ i \ odd, D_i \to b \ if \ i \geq 2 \ and \ i \ even, C_i \to x\}.$

For$i > 1$ and i even
$R^{h_i} = \{S_i \to A_iX, X \to B_iX, X \to C_i$
$R^{v_i} = \{A_i \to xA_i, A_i \to x, B_i \to bD_i, D_i \to .D_i, C_i \to xC_i, \}.$

For$i > 1$ and i odd
$R^{h_i} = \{S_i \to A_iX, X \to B_iX, X \to C_i$
$R^{v_i} = \{A_i \to xA_i, A_i \to x, B_i \to aD_i, D_i \to .D_i, C_i \to xC_i, \}.$

$$M = \{ \begin{array}{|c|}\hline a \\ \hline . \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array}$$
$$\begin{array}{|c|}\hline b \\ \hline . \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array}$$
$$\begin{array}{|c|}\hline . \\ \hline . \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline x \\ \hline\end{array}$$
$$\begin{array}{|c|}\hline . \\ \hline D_i \\ \hline\end{array} \ \# \ \begin{array}{|c|}\hline x \\ \hline C_i \\ \hline\end{array} \ \$ \ \# \ \begin{array}{|c|}\hline x \\ \hline A_i \\ \hline\end{array} \}$$

We note that the top and bottom rows of the rectangular arrays generated in the individual language will be of the form $xa^mxb^mxa^mxb^m...x$ as there are n component grammars.

Theorem 2 For $Y \in \{I, T\}$,

1. $2DRML = Y_{isgs}L_1(REG) \subset Y_{isgs}L_2(REG) \subset \cdots \subset Y_{isgs}L_n(REG) \subset \cdots$

2. $2DCFML = Y_{isgs}L_1(CF) \subset Y_{isgs}L_2(CF) \subset \cdots \subset Y_{isgs}L_n(CF) \subset \cdots$

The first statement follows in view of the Example 4. The second can be shown on similar lines.

4 Conclusion

The image splicing grammar system introduced in this paper appears to be a powerful means of generating picture arrays. It remains to compare other picture generating mechanisms with these systems.

References

[1] E. Csuhaj-Varjú, Grammar systems: 12 years, 12 problems (short version), In R. Freund and A. Kelemenová (Eds.), Proceedings of the International Workshop on Grammar Systems 2000, 2000, 77-92, Silesian University, Opava.

[2] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun, Grammar systems: A grammatical approach to distribution and cooperation, Gordon and Breach Science Publishers, 1994.

[3] J. Dassow and V. Mitrana, Splicing grammar systems, Computers and Artificial Intelligence, 15, 1996, 109-122.

[4] R. Freund, Array Grammar Systems, Journal of Automata, Languages and Combinatorics 5,1, 2000, 13-29.

[5] D. Giammarresi and A. Restivo, Two-dimensional languages, In "Handbook of Formal Languages" Vol.3, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, 1997, 215-267.

[6] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, Bull. Math. Biology, 49. 1987, 737-759.

[7] T. Head, Gh. Păun and D. Pixton, Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination, In "Handbook of Formal Languages" Vol.2, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, 1997, 295 - 360.

[8] P. Helen Chandra, K.G. Subramanian, and D.G. Thomas, Parallel Splicing on Images, To appear in Int. J. of pattern recognition and artificial intelligence.

[9] A. Rosenfeld and R. Siromoney, Picture languages - a survey, Languages of design, 1, 1993, 229–245.

[10] G. Siromoney, R. Siromoney and K. Krithivasan, Abstract families of matrices and picture languages, Computer Graphics and Image Processing, 1, 1972, 234-307.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 287 - 299, 2004.

Distributed Probabilistic Finite Automata

G. N. Santhana Krishnan, Kamala Krithivasan, Ashish Choudhary

Dept of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India 600036.
kamala@iitm.ernet.in

Abstract

In this paper we consider multiple choice probabilistic automaton and show that it can be simulated by a (single choice) probabilistic automaton. We define distributed probabilistic automata with four cooperating modes and show that distribution does not give any additional power.

1 Introduction

Recent developments in the field of Computer Science are towards processing information that are distributed among geologically different locations. In various fields like Artificial Intelligence, Cognitive Psychology etc, we have to deal with more and more complex tasks distributed among set of 'processors', which are working together in a well defined way. Parallel computers, computer nets, distributed databases and knowledge sources are practical materialization of this idea. Formal Language Theory constructs, namely Grammar Systems, modelling distribution and parallelism were coined in the early nineties. [1] deals with many types of Grammar Systems. Distributed Automata have been considered in [2].

Very often, in the world of distributed information processing systems, there arise situations where the system is random and its course of operation nondeterministic. Probabilistic grammars were introduced as mathematical generative models for capturing the randomness in a classical computing environment [3, 4]. Probabilistic grammar systems were studied in [5, 6] with application to network-load modelling [7].

In this paper we study distributed probabilistic automata. We first study multiple choice probabilistic automaton where we include the idea of both randomness and non-determinism. We show that a multiple choice probabilistic automaton can be simulated by a single choice probabilistic automaton. We define distributed probabilistic automata and consider four standard modes of cooperation between the components, viz, $*, = k, \leq k, \geq k$ modes. We show that a distributed probabilistic automaton can be simulated by a multiple choice probabilistic automaton. Since multiple choice probabilistic automaton can be simulated by a single choice probabilistic automaton, we see that distribution does not increase the power of probabilistic automaton in any of the four modes of cooperation.

In the next section we define probabilistic automata. In section 3 we define multiple choice probabilistic automata and show the equivalence to single choice probabilistic automata. In section 4 we define distributed probabilistic automaton with four modes of cooperation and show that in any mode, it can be simulated by a multiple choice probabilistic automaton. The paper concludes with some remarks in section 5.

2 Probabilistic Automaton

In this section, we consider the definition of probabilistic automaton. For the definition and details of probabilistic grammars, the reader is referred to [3, 4]. A study of probabilistic automata is done in [8].

Definition 2.1 A finite probabilistic automaton over a finite alphabet V is an ordered triple $\mathbf{PA} = (S, s_0, M)$ where $S = \{s_1, s_2, s_3, ..., s_n\}$ is a finite set with $n \geq 1$ elements(the set of internal states), s_0 is an n-dimensional stochastic row vector(the initial distribution) and M is a mapping of V into the set of n-dimensional stochastic matrices. For $x \in V$, the $(i, j)^{th}$ entry in the matrix $M(x)$ is denoted by $p_j(s_i, x)$ and referred to as the transient probability of **PA** to enter into the state s_j, after being in the state s_i after consuming the input x.

As an example, consider the following example. $\mathbf{PA}_1 = (\{s_1, s_2\}, (1, 0), M)$ over the alphabet $\{x, y\}$ where

$$M(x) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad M(y) = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

The initial distribution indicates that s_1 is the initial state. From the matrices M we see that s_1 changes to s_2 with a probability of $1/2$ on reading the symbol y. This can be indicated in a diagram with the states being the nodes and arcs having labels in the form $x(p)$ where x is the symbol while p is the probability of transition from one node to the other on scanning the symbol x.

For a finite probabilistic automaton, we increase the domain of M from V to V^* as follows:

1. $M(\epsilon) = I$
2. $M(wx) = M(w)M(x)$

Now for a word w, the $(i, j)^{th}$ entry of $M(w)$ would denote the probability that the automaton would move to state s_j after getting the input w if it were initially in state s_i.

Definition 2.2 Let $\mathbf{PA} = (\{s_1, \ldots, s_n\}, s_0, M)$ be a finite probabilistic automaton over V and $w \in V^*$. The stochastic row vector $s_0M(w)$ is termed as the distribution of states caused by the word w and is denoted by $\mathbf{PA}(w)$.

We notice that for a word w, the ith entry of $\mathbf{PA}(w)$ is the probability that the automaton is in state s_i starting from the initial distribution s_0.

Definition 2.3 Let $\overline{s}_1$ be a $n-$dimensional column vector, each component of which equals either 0 or 1, and the **PA** as in the previous definition. Let η be a real number such that $0 \leq \eta \leq 1$. The language accepted in **PA** by $\overline{s}_1$ with the cut-point η is defined by

$$\mathrm{L}(\mathbf{PA}, \overline{s_1}, \eta) = \{w | s_0 M(w) \overline{s_1} \geq \eta\}.$$

A language L is $\eta-$stochastic if for some **PA**, η and $\overline{s_1}$, $L = L(\mathbf{PA}, \overline{s_1}, \eta)$. A language L is stochastic if it is $\eta-$stochastic for some η.

Theorem 2.1 *Every regular language is stochastic. Further, every regular language is $\eta-$stochastic for every $0 \leq \eta \leq 1$.*

Theorem 2.2 *Every 0-stochastic language is regular.*

3 Multiple-Choice Probabilistic Automaton

Now we consider multiple choice probabilistic automaton.

Definition 3.1 (Multiple-Choice Probabilistic Automaton) A finite Multiple-Choice probabilistic Automaton over a finite alphabet V is an ordered triple **NDPA** $= (S, s_0, M)$ where $S = \{s_1, s_2, \ldots, s_n\}$ is a finite set with $n \geq 1$ elements (the set of internal states), s_0 is an $n-$dimensional stochastic row vector(the initial distribution) and M is a collection of mapping of V into the set of $n-$dimensional stochastic matrices. For $x \in V$, the $(i, j)^{th}$ entry in each of the matrices in $M(x)$ is denoted by $p_j(s_i, x)$ and referred to as the transient probability of **NDPA** to enter into the state s_j after being in the state s_i and consuming the input x.

We define the language accepted in a similar fashion as above, with the consideration that whenever a matrix needs to be chosen for an alphabet symbol a, the choice is made non-deterministically among the matrices in $M(a)$. We talk about two different modes of acceptance, namely the $max-$mode and the $p-$mode.

Definition 3.2 $L_{max}(\mathbf{NPA}, \overline{s_1}, \eta) = \{w | max\{s_0 M(w) \overline{s_1}\} \geq \eta\}$, where $\overline{s_1}$, η and s_0 are as defined above and $\{s_0 M(w) \overline{s_1}\}$ denotes the set of all derivations of the string w.

Definition 3.3 $L_p(\mathbf{NPA}, \overline{s_1}, \eta) = \{w | prob\{s_0 M(w) \overline{s_1}\} \geq \eta\}$, where $\overline{s_1}$, η and s_0 are as defined above and $prob\{s_0 M(w) \overline{s_1}\}$ denotes the probability of the derivation of the string w considering all possible derivations of the string w.

Let $L_{max}(\mathbf{NPA})$ denote the family of languages generated by multiple choice probabilistic automata in the $max-$ mode, $L_p(\mathbf{NPA})$ denote the family of languages generated in the $p-$mode and $L(\mathbf{PA})$ denote the family of languages generated by probabilistic automata, i.e. let it denote the family of stochastic languages.

Theorem 3.1 $L_p(\mathbf{NPA}) = L(\mathbf{PA})$.

Proof. Let $M_1(x), M_2(x), \ldots, M_k(x)$ be the various choices for the alphabet x. Let the system be in distribution $\overline{s_1}$, and the input which is read next is x. The probability that the new state is s_j is :

$$p_j = \sum_{l=1}^{k} p(l) * (\sum_{i=0}^{n} \overline{s_{1i}} * p_{lj}(s_i, x))$$

Where

- p_j is the probability that the automaton is in state s_j after reading the alphabet x.
- $\overline{s_{1i}}$ is the ith entry in $\overline{s_1}$. In other word, it stands for the probability that the system were in the state s_i initially.
- $p_{lj}(s_i, x))$ is the transient probability in $M_l(x)$ that the automaton would change from state s_i to s_j on encountering the symbol x.
- $p(l)$ is the probability of choice of the matrix $M_l(x)$.

Noting that the matrices are chosen non-deterministically, the above equation reduces to

$$p_j = \sum_{l=1}^{k} (1/k) * (\sum_{i=0}^{n} \overline{s_{1i}} * p_{lj}(s_i, x))$$

Now the above can be written in the vector notation as

$$\overline{s_2} = \sum_{l=1}^{k} (1/k) * \overline{s_1} * M_l(x)$$

or in other words,

$$\overline{s_2} = \overline{s_1} * \sum_{l=1}^{k} (1/k) * M_l(x)$$

If we consider the **PA** with (S, s_0, M), with the set of all matrices for each alphabet replaced with their arithmetic mean, we obtain the same distribution of states at each stage. So, this clearly proves that Multiple-Choice probabilistic automata are at least as powerful as probabilistic automata. Also, since every probabilistic automaton is also a multiple-choice probabilistic automaton, the inclusion holds the other way too. Hence the theorem. □

4 Distributed Probabilistic Automaton

We now define the various kinds of distributed probabilistic automaton with certain restrictions as to transfer of control, and we study various families of languages accepted.

Definition 4.1 (**Distributed Probabilistic Automaton**) A distributed finite probabilistic automaton over a finite alphabet V is an ordered $(k+2)$-tuple **DPA** $= (S, s_0, M_1, M_2, \ldots, M_k)$, where $S = \{s_1, s_2, \ldots, s_n\}$ is a finite set with $n \geq 1$ elements(the set of internal states), s_0 is an $n-$dimensional stochastic row vector(the initial distribution) and each of the M_i's is mapping of V into the set of $n-$dimensional stochastic matrices. For $x \in V$, the $(j,k)^{th}$ entry of each of the matrices in $M_i(x)$ is denoted by $p_{ik}(s_j, x)$ and referred to as the transient probability of **DPA** to enter into the state s_k, after being in the state s_j after consuming the input x in component i.

We now define the various modes of cooperation. To achieve that end, we first define the various modes of transition.

Definition 4.2 (**Transition Modes**) There are four possible modes of transitions.

- $*$-mode: The transition from M_i to some other component M_j occurs at any arbitrary stage.
- =k-mode: The transition from M_i to some other component M_j occurs after exactly k transitions in M_i.
- $\geq k$-mode: The transition from M_i to some other component M_j occurs after at least k transitions in M_i.
- $\leq k$-mode: The transition from M_i to some other component M_j occurs before k+1 transitions in M_i.

We note that conventional $t-$mode does not make sense in this context since all the matrices involved are stochastic, and hence the derivation in any M_i cannot terminate unless the string has been completely processed. Also the derivation always starts with M_1 (without loss of generality, this can be assumed).

Definition 4.3 $L_{(p,\alpha)}(DPA, \eta) = \{w \in V^* | prob\{s_0 M(w)\overline{s_1}\} \geq \eta\}$, where α is the transition mode, $\alpha \in \{*, = k, \leq k, \geq k | k \geq 1\}$ and $prob\{s_0 M(w)\overline{s_1}\}$ denotes the probability of derivation of the string w considering all possible derivations of the string w.

Definition 4.4 $L_{(max,\alpha)}(DPA, \eta) = \{w \in V^* | max\{s_0 M(w)\overline{s_1}\} \geq \eta\}$, where α is the transition mode, $\alpha \in \{*, = k, \leq k, \geq k | k \geq 1\}$ and $\max\{s_0 M(w)\overline{s_1}\}$ refers to the maximum probability of the derivation of the string w amongst all the possible derivations of it.

We study the acceptance power of **DPA** in the various modes of transition. We prove that $L_{(p,\alpha)}$ generates just a stochastic language in the following theorem. We leave the study of $L_{(max,\alpha)}$ as an open problem.

Theorem 4.1 *Given* $\alpha \in \{*, = k, \leq k, \geq k\}$ *and given a* **DPA M**, *there exists a* **PA M**$''$ *such that* $L_{(p,\alpha)}(s_0, M, \eta) = L(s_0, M'', \eta)$.

Proof. **Case 1:** $\alpha = *$.

Let $\mathbf{M} = (S, s_0, M_1, M_2, \ldots, M_l)$. Let the cardinality of the set S is n. We try to define a corresponding **PA** $\mathbf{M}'' = (S'', s_0'', M'')$ in the following way.

$S'' = \{[q, i] | 1 \leq i \leq l, q \in S\}$. Without loss of generality, we assume that the machine starts working in the first component. s_0'' is a nl stochastic row vector, which defines the new initial state distribution in $\mathbf{M}''$. Since we have assumed that the machine $\mathbf{M}$ starts working in the first component, so if the initial state distribution in $\mathbf{M}$ is $(p_{q_1}, p_{q_2}, \ldots, p_{q_n})$, then the initial state distribution in machine $\mathbf{M}''$ would be represented as:

$$s_0'' = (p_{[q_1,1]}, p_{[q_2,1]}, \ldots, p_{[q_n,1]}, 0, 0, \ldots, 0(nl - n)times)$$

At any instance of time, the machine $\mathbf{M}$ can be in any one of the component. Suppose at current instance t, the machine $\mathbf{M}$ is in component i, then the state distribution in $\mathbf{M}$ would have been represented as:

$$s_t = (p_{q_1}, p_{q_2}, \ldots, p_{q_n})$$

The corresponding state configuration in machine $\mathbf{M}''$ would be represented by a nl stochastic row vector as:

$$s_t'' = (0, 0, \ldots, p_{[q_1,i]}, p_{[q_2,i]}, \ldots, p_{[q_n,i]}, 0, 0, \ldots, 0)$$

where the non-zero entries are from the position $n(i-1)+1$ to $n(i-1)+n$ and $p_{[q_j,i]} = p_{q_j}$, $\forall 1 \leq j \leq n$. Also the final state configuration has to be changed in $\mathbf{M}''$. If the final state configuration in $\mathbf{M}$ is $(p_{q_1}, p_{q_2}, \ldots, p_{q_n})^T$ where each of the values p_{q_i}, $\forall i, 1 \leq i \leq n$ is either 0 or 1, then the corresponding final state configuration in $\mathbf{M}''$ would be represented by a nl-dimensional column vector:

$$(p_{[q_1,1]}, p_{[q_2,1]}, \ldots, p_{[q_n,1]}, p_{[q_1,2]}, p_{[q_2,2]}, \ldots, p_{[q_n,2]}, \ldots, p_{[q_1,l]}, p_{[q_2,l]}, \ldots, p_{[q_n,l]})^T$$

where $p_{[q_i,j]} = p_{q_i}$, $\forall i, j, 1 \leq i \leq n, 1 \leq j \leq l$. From each of the M_i, we have l possible matrices to choose and from each of the matrices, we choose the matrix for given alphabet. We can encompass both these steps into a single step. The change of M_i to M_j can be done by having the present stochastic vector, representing the current state distribution, multiplied by the following matrix:

$$M(i,j) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where M is a (nl)x(nl) matrix and I is a nxn identity matrix which starts off at the entry $(i-1)n+1$, $(j-1)n+1$. It can be easily seen that the matrix carries the transition from M_i to M_j. For example, consider $M(1,2)$. This is a matrix which carries out the transformation from M_1 to M_2. Considering that $n = 2, l = 3$ we have:

$$M(1,2) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose the state were $\overline{s} = (p_1, p_2, 0, 0, 0, 0)$.

Then $\overline{s_1} = \overline{s} * M(1,2)$, which is

$$(p_1, p_2, 0, 0, 0, 0) * \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which is $(0, 0, p_1, p_2, 0, 0)$.

We thus see that $M(i,j)$ transfers control from M_i to M_j.

For each alphabet a, we define the matrix as follows:

$$M''(a) = \begin{bmatrix} M_1(a) & 0 & . & 0 \\ 0 & M_2(a) & . & 0 \\ . & . & . & . \\ . & 0 & . & M_l(a) \end{bmatrix}$$

Consider the same example as above. Suppose, we have $l = 3$ and $n = 2$. Suppose, the present vector is $(p_1, p_2, 0, 0, 0, 0)$, and we read the symbol a. The matrix for a would be

$$M''(a) = \begin{bmatrix} p_{11}(s_1,a) & p_{12}(s_1,a) & 0 & 0 & 0 & 0 \\ p_{11}(s_2,a) & p_{12}(s_2,a) & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{21}(s_1,a) & p_{22}(s_1,a) & 0 & 0 \\ 0 & 0 & p_{21}(s_2,a) & p_{22}(s_2,a) & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{31}(s_1,a) & p_{32}(s_1,a) \\ 0 & 0 & 0 & 0 & p_{31}(s_2,a) & p_{32}(s_2,a) \end{bmatrix}$$

Now,consider $\overline{s_2}$.

$\overline{s_2} = \overline{s} * M''(a)$, which is

$$(p_1,p_2,0,0,0,0)* \begin{bmatrix} p_{11}(s_1,a) & p_{12}(s_1,a) & 0 & 0 & 0 & 0 \\ p_{11}(s_2,a) & p_{12}(s_2,a) & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{21}(s_1,a) & p_{22}(s_1,a) & 0 & 0 \\ 0 & 0 & p_{21}(s_2,a) & p_{22}(s_2,a) & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{31}(s_1,a) & p_{32}(s_1,a) \\ 0 & 0 & 0 & 0 & p_{31}(s_2,a) & p_{32}(s_2,a) \end{bmatrix}$$

which is $(p_1 * p_{11}(s_1,a) + p_2 * p_{11}(s_2,a), p_1 * p12(s_1,a) + p_2 * p_{12}(s_2,a), 0, 0, 0, 0)$.

We can view it in other way. The above can be seen as follows in the block matrix-multiplication format.

$$\begin{bmatrix} s_{n,1} * M_1(a) & s_{n,2} * 0 & s_{n,3} * 0 \\ s_{n,1} * 0 & s_{n,2} * M_2(a) & s_{n,3} * 0 \\ s_{n,1} * 0 & s_{n,2} * 0 & s_{n,3} * M_3(a) \end{bmatrix}$$

where $s_{n,1}, s_{n,2}$ and $s_{n,3}$ represent the first, second and third n elements in the current state configuration. In the context of the above example, $s_{n,1}, s_{n,2}$ and $s_{n,3}$

represents the first, second and last two elements in the current state configuration. So, we now see that $M''(a)$ helps us process the string as if it were under the control of one M_i.

In order to incorporate the transition from M_i to M_j, we premultiply $M''(a)$ by $M(i,j)$.

Consider the above example. Instead of having just $M''(a)$, we will have $M(1,2)*M''(a)$, $M(1,3)*M''(a)$, $M(2,1)*M''(a)$, $M(2,3)*M''(a)$, $M(3,1)*M''(a)$, $M(3,2)*M''(a)$ and of course $M''(a)$ itself. For each alphabet, thus we have $l*(l-1)+1$ matrices to choose from. We are now left with the $p-$mode of acceptance in the multiple-choice probabilistic automaton, which we have already proved to be equivalent to probabilistic automaton.

Case 2: $\alpha = =$k.

Suppose we are given a distributed probabilistic automaton $\mathbf{M}$ with l components and n states

$$\mathbf{M} = (S, s_0, M_1, M_2, M_3, \ldots, M_l)$$

Without loss of generality, we assume that initially the automaton is in the first component. Since the mode of cooperation is $= k$, the automaton should spend exactly k steps in the current component before switching to some other component, other than the present component.

To simulate the working of this distributed automaton with a single probabilistic automaton, we look at $\mathbf{M}$ in a different way. We think that we have exactly k copies of each of the components $M_1, M_2, M_3, \ldots, M_l$ i.e

$$M_{11}, M_{12}, M_{13}, \ldots, M_{1k}$$

$$M_{21}, M_{22}, M_{23}, \ldots, M_{2k}$$

$$\ldots$$

$$M_{l1}, M_{l2}, M_{l3}, \ldots, M_{lk}$$

Here $M_{ij} = M_i$, $\forall i, j$, $1 \leq i \leq l, 1 \leq j \leq k$.

All the transition matrices in $M_{11}, M_{12}, M_{13}, \ldots, M_{1k}$ are identical as in M_1. Similarly all the transition matrices in $M_{21}, M_{22}, M_{23}, \ldots, M_{2k}$ are identical as in M_2 and so on. Now the original distributed probabilistic automaton $\mathbf{M}$ is converted into another distributed probabilistic automaton $\mathbf{M}'$ as follows:

$$\mathbf{M}' = (S', s'_0, M_{11}, M_{12}, \ldots, M_{1k}, M_{21}, M_{22}, \ldots, M_{2k}, \ldots, M_{l1}, M_{l2}, \ldots, M_{lk})$$

where

- S' is the new set of states in $\mathbf{M}'$ such that $S' = \{[q, ij] | 1 \leq i \leq l, 1 \leq j \leq k, q \in S\}$. In $\mathbf{M}'$ since the components are replicated, we also include the number of the component and the number of the copy of the component to which the state belongs along with each state in the new set of states. The cardinality of the set S' is nlk.

- s'_0 is a $n*l*k$ stochastic row vector which denotes the new initial probability distribution of $\mathbf{M}'$. If $s_0 = (p_{q_1}, p_{q_2}, \ldots, p_{q_n})$ then we have :

$$s'_0 = (p_{[q_1,11]}, p_{[q_2,11]}, \ldots, p_{[q_n,11]}, 0, 0, \ldots, 0(n*l*k-n)times)$$

  such that $p_{q_i} = p_{[q_i,11]}$ $\forall i, 1 \leq i \leq n$. Here p_{q_j} is interpreted as the probability of being in state q_j in $\mathbf{M}$ and $p_{[q_j,11]}$ is interpreted as the probability of being in the state $[q_j, 11]$ in $\mathbf{M}'$.

- $M_{11}, M_{12}, \ldots, M_{1k}, M_{21}, M_{22}, \ldots, M_{2k}, \ldots, M_{l1}, M_{l2}, \ldots, M_{lk}$ are the set of transition matrices such that :

$$M_{i1} = M_{i2} = M_{i3} = \ldots, = M_{ik} = M_i, \forall i, 1 \leq i \leq l.$$

The probabilistic state distribution at any instance of time is given by a $n*l*k$ stochastic row vector such that if at current instance the M_{ij}^{th} component is active then the probability distribution is:

$$(0, 0, \ldots, 0, p_{[q_1,ij]}, p_{[q_2,ij]}, \ldots, p_{[q_n,ij]}, 0, 0, \ldots, 0)$$

where the non-zero entries are from the position $[(i-1)*k+j-1]*n+1$ to $[(i-1)k+j-1]*n+n$.

In the original **DPA M**, if the system is in some component M_i, then it remains in M_i for exactly k steps and after that it non-deterministically switches to some other component M_j, $1 \leq j \leq l, j \neq i$ for next k steps. This step is simulated in $\mathbf{M}'$ by forcing the transition from M_{i1} to M_{i2}, from M_{i2} to M_{i3},..., from $M_{i(k-1)}$ to M_{ik} and then from M_{ik} to M_{j1} , $1 \leq j \leq l$, $j \neq i$. Also $\mathbf{M}'$ makes sure that the system spends only one step in any component M_{ij}, $1 \leq i \leq l, 1 \leq j \leq k$ at any instance of time. The change of component from M_{ij} to $M_{ij(+1)}$, $\forall i, j$, $1 \leq i \leq l, 1 \leq j < k$ is done in $\mathbf{M}'$ by having the present stochastic row vector representing the current state distribution multiplied by the following matrix:

$$M(ij, i(j+1)) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where $M(ij, i(j+1))$ is a $(n*l*k)$ x $(n*l*k)$ matrix and I is a $n*n$ identity matrix staring at the entry $[(i-1)k+j-1]n+1$, $[(i-1)k+j]n+1$.

The change from the component M_{ik} to M_{j1}, $\forall i, j$, $1 \leq i \leq l, 1 \leq j \leq l, i \neq j$ is done in $\mathbf{M}'$ by having the present stochastic row vector representing the current state distribution multiplied by the following matrix:

$$M(ik, j1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where $M(ik, j1)$ is a $(n*l*k)$ x $(n*l*k)$ matrix and I is a $n*n$ identity matrix staring at the entry $(ik-1)n+1$, $[(j-1)]n+1$.

To understand the above notions, we take an example. Suppose we have $l = 3, k = 4, n = 2$. Then in $\mathbf{M}'$ the state configuration is represented by a 1 x 24 stochastic row vector. Suppose that at current instance the machine $\mathbf{M}'$ is in the component M_{22}, then the present configuration is :

$$\overline{s_1} = (0,0,0,0,0,0,0,0,0,0,p_{[q_1,22]},p_{[q_2,22]},0,0,0,0,0,0,0,0,0,0,0,0)$$

To switch the current component from M_{22} to M_{23} we multiply $\overline{s_1}$ with the following matrix:

$$M(22,23)=\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

Multiplying $\overline{s_1}$ with $M(22,23)$ we get new state distribution as :

$$\overline{s_2} = (0,0,0,0,0,0,0,0,0,0,0,0,p_{[q_1,23]},p_{[q_2,23]},0,0,0,0,0,0,0,0,0,0)$$

such that $p_{[q_1,23]} = p_{[q_1,22]}$ and $p_{[q_2,23]} = p_{[q_2,22]}$.

Now we construct an equivalent probabilistic automata $\mathbf{M}''$ having single component with multiple choice for each alphabet symbol from $\mathbf{M}'$ as follows:

$$\mathbf{M}'' = (S'', s_0'', M'')$$

where

- $S'' = S'$.

- $s_0'' = s_0'$.
- M'' is the set of transition matrices.

The final state configuration in $\mathbf{M}''$ has to be changed. If the final state configuration in $\mathbf{M}$ is $(p_{q_1}, p_{q_2}, \ldots, p_{q_n})^T$, then the corresponding final state configuration in $\mathbf{M}''$ is represented as a nlk-dimensional column vector
$(0, 0, \ldots, 0(nk-n)times, p_{[q_1,1k]}, p_{[q_2,1k]}, \ldots, p_{[q_n,1k]}, 0, 0, \ldots, 0(nk-n)times, p_{[q_1,2k]}, p_{[q_2,2k]}, \ldots, p_{[q_n,2k]}, \ldots, 0, 0 \ldots, 0(nk-n)times, p_{[q_1,lk]}, p_{[q_2,lk]}, \ldots, p_{[q_n,lk]})^T$
where $p_{[q_i,jk]} = p_{q_i}$, $\forall i, j, 1 \leq i \leq n, 1 \leq j \leq l$. The matrices in the set M'' is constructed from the matrices $M_{11}, M_{12}, \ldots, M_{lk}$ as follows:

For each alphabet a, we define the transition matrix $M''(a)$ as follows:

$$M''(a) = \begin{bmatrix} M_{11}(a) & & & & & & & & & & & \\ & M_{12}(a) & & & & & & & & & & \\ & & \cdot & & & & & & & & & \\ & & & M_{21}(a) & & & & & & & & \\ & & & & M_{22}(a) & & & & & & & \\ & & & & & \cdot & & & & & & \\ & & & & & & M_{2k}(a) & & & & & \\ & & & & & & & \cdot & & & & \\ & & & & & & & & M_{l1}(a) & & & \\ & & & & & & & & & M_{l2}(a) & & \\ & & & & & & & & & & \cdot & \\ & & & & & & & & & & & M_{lk}(a) \end{bmatrix}$$

Here $M_{ij}(a) = M_i(a)$, $\forall i, j$, $1 \leq i \leq l, 1 \leq j \leq k$.

Example: Suppose we have $l = 2, k = 2, n = 2$, then $M^{\text{“}}(a)$ is defined as:

$$\begin{bmatrix} p_{[11,1]}(s_1,a) & p_{[11,2]}(s_1,a) & 0 & 0 & 0 & 0 & 0 & 0 \\ p_{[11,1]}(s_2,a) & p_{[11,2]}(s_2,a) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{[12,1]}(s_1,a) & p_{[12,2]}(s_1,a) & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{[12,1]}(s_2,a) & p_{[12,2]}(s_2,a) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{[21,1]}(s_1,a) & p_{[21,2]}(s_1,a) & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{[21,1]}(s_2,a) & p_{[21,2]}(s_2,a) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{[22,1]}(s_1,a) & p_{[22,2]}(s_1,a) \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{[22,1]}(s_2,a) & p_{[22,2]}(s_2,a) \end{bmatrix}$$

where

$$\begin{bmatrix} p_{[11,1]}(s_1,a) & p_{[11,2]}(s_1,a) \\ p_{[11,1]}(s_2,a) & p_{[11,2]}(s_2,a) \end{bmatrix} = \begin{bmatrix} p_{[12,1]}(s_1,a) & p_{[12,2]}(s_1,a) \\ p_{[12,1]}(s_2,a) & p_{[12,2]}(s_2,a) \end{bmatrix} = M_1(a)$$

Similarly

$$\begin{bmatrix} p_{[21,1]}(s_1,a) & p_{[21,2]}(s_1,a) \\ p_{[21,1]}(s_2,a) & p_{[21,2]}(s_2,a) \end{bmatrix} = \begin{bmatrix} p_{[22,1]}(s_1,a) & p_{[22,2]}(s_1,a) \\ p_{[22,1]}(s_2,a) & p_{[22,2]}(s_2,a) \end{bmatrix} = M_2(a)$$

Now as in the case of $*$-mode, we will premultiply $M''(a)$ with the matrices which forces $\mathbf{M}''$ to switch the component from M_{ij} to $M_{i(j+1)}$ if $j < k$ or otherwise from M_{ik} to M_{j1} , $\forall 1 \leq i \leq l, 1 \leq j \leq l, i \neq j$.

Example: In the above example, we will have $M(11,12)(a) * M''(a)$, $M(21,22)(a) * M''(a)$, $M(12,21)(a) * M''(a)$, $M(22,11)(a) * M''(a)$.

So now in $\mathbf{M}''$ we have $l(k-1)+l(l-1)$ matrices in total for each alphabet symbol a. Thus from the distributed probabilistic automaton $\mathbf{M}$ we get a multiple choice probabilistic automaton $\mathbf{M}''$, and we have already proved that any multiple choice probabilistic automaton can be simulated by a probabilistic automaton(with single choice). Thus a distributed probabilistic automaton with $= k$-mode of cooperation in p-mode of acceptance is equivalent to a probabilistic automaton.

Case 3: $\alpha = \leq k$ **.**

The idea of the proof in this case will be same as that of $= k$-mode. Suppose we are given a distributed probabilistic automaton with l components

$$\mathbf{M} = (S, s_0, M_1, M_2, \ldots, M_l)$$

We construct $\mathbf{M}'$ from $\mathbf{M}$ and $\mathbf{M}''$ from $\mathbf{M}'$ in the same way as in the case of $= k$-mode. The final state configuration here has to be changed as it is changed in $= k$-mode, with slight difference. If the final state distribution in $\mathbf{M}$ is $(p_{q_1}, p_{q_2}, \ldots, p_{q_n})^T$, then the corresponding final state configuration in $\mathbf{M}''$ is represented by a nlk-dimensional column vector:
$(p_{[q_1,11]}, p_{[q_2,11]}, \ldots, p_{[q_n,1k]}, p_{[q_1,12]}, p[q_2,12], \ldots, p_{[q_n,12]}, \ldots, p_{[q_1,lk]}, p_{[q_2,lk]}, \ldots,$
$p_{[q_n,lk]})^T$, where $p_{q_i} = p_{[q_i,jm]}$, $\forall i,j,m, 1 \leq i \leq n, 1 \leq j \leq l, 1 \leq m \leq k$. The only difference here occurs in the choice of matrices in $\mathbf{M}''$. Since the machine $\mathbf{M}$ is working in $\leq k$-mode, the system remains in any component at most k steps after which it has to forcibly switch to some other component. This is simulated in $\mathbf{M}''$ by forcing the machine to make the transition from M_{ij} to $M_{i(j+1)}$ or M_{r1}, $\forall i,j,r : 1 \leq i \leq l, j < k, 1 \leq r \leq l, r \neq i$ or from M_{ik} to M_{r1} , $\forall i,k,r : 1 \leq i \leq l, 1 \leq r \leq l, r \neq i$.

Example : Considering the same example as in $= k$-mode with $l = 2, k = 2, n = 2$, in $\mathbf{M}''$ we will have $M(11,12) * M''(a)$, $M(11,21) * M''(a)$, $M(12,21) * M''(a)$, $M(21,22) * M''(a)$, $M(21,11) * M''(a)$, $M(22,11) * M''(a)$.

So, for each alphabet symbol a we will now have $ll(k-1) + l(l-1)$ matrices in total in $\mathbf{M}''$. Thus the distributed probabilistic automaton $\mathbf{M}$ is reduced to a multiple choice probabilistic automaton $\mathbf{M}''$ and we have proved that any multiple choice probabilistic automaton is equivalent to a probabilistic automaton with single choice for each alphabet symbol. Hence the theorem holds for the $\leq k$-mode also.

Case 4: $\alpha = \geq k$**.**

The proof is similar to the $\leq k$-mode. Suppose we are given a distributed probabilistic automaton with l components

$$M = (S, s_0, M_1, M_2, \ldots, M_l)$$

We construct $\mathbf{M}'$ from $\mathbf{M}$ and $\mathbf{M}''$ from $\mathbf{M}'$ in the same way as in the case of $= k$-mode. The only difference here occurs in the choice of matrices in $\mathbf{M}''$. Since the machine $\mathbf{M}$ is working in $\geq k$-mode, the machine spends at least k steps in each component before switching the component. This is simulated in $\mathbf{M}''$ by forcing the transition from M_{ij} to $M_{i(j+1)}$, $\forall i,j, 1 \leq i \leq l, j < k$ or otherwise from M_{ik} to M_{ik} or M_{r1}, $\forall i,r, 1 \leq i \leq l, 1 \leq r \leq l, r \neq i$. Also the final state configuration has to be changed exactly in the same way as in the case of $= k$-mode.

Example: Considering the same example as in $\geq k$-mode with $l = 2, k = 2, n = 2$, we will have $M(11,12) * M''(a)$, $M(12,12) * M''(a)$, $M(12,21) * M''(a)$, $M(21,22) * M''(a)$, $M(22,22) * M''(a)$, $M(22,11) * M''(a)$.

So, in general for each alphabet symbol a we will have $l(k-1) + l + l(l-1)$ matrices in $\mathbf{M}''$. Thus the original distributed probabilistic automaton is reduced to a multiple choice probabilistic automaton $\mathbf{M}''$ and we have shown that any multiple choice probabilistic automaton is equivalent to a probabilistic automaton with single choice for each alphabet symbol. Hence the theorem holds for the case $\alpha = \geq k$-mode also.

5 Conclusion

In this paper, we have considered multiple choice probabilistic automaton and we have shown that it can be simulated by a (single choice) probabilistic automaton. We defined distributed probabilistic automaton with four cooperating modes. We have shown that in each of the modes, a distributed probabilistic automaton can be simulated by a multiple choice probabilistic automaton and hence by a (single choice) probabilistic automaton. Here we have not considered probability of transition between components, which can also be taken into account and the above results can be established with appropriate modifications in the proofs.

References

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

[2] K. Krithivasan, M. Sakthi Balan and P. Harsha, Distributed processing in automata, *International Journal of Foundation of Computer Science*, 10(4), p 443-464, 1999.

[3] K. S. Fu, Stochastic automata, stochastic languages and pattern recognition, *IEEE Symposium on Decision Control*, p 31-49, 1970.

[4] A. Salomaa, Probabilistic and weighted grammars, *Information and Control*, 15, p 529-544, 1969.

[5] K. Arthi and K. Krithivasan, Probabilistic parallel communicating grammar systems, *International Journal of Computer Mathematics*, 79(1) p 1-26, 2002.

[6] K. Arthi, K. Krithivasan and E. Csuhaj-Varjú, On the rule number of complexity of components of probabilistic cooperating grammar system, *JALC*, 7, p 433-446, 2002.

[7] K. Arthi, K. Krithivasan and S. V. Raghvan, A generative model for capturing user behaviour in computer networks, *Proceedings of SCI 2001*, Vol 5, p 162-167, 2001.

[8] Azaria Paz. Introduction to Probabilistic Automata, Academic Press, 1971.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 300 - 312, 2004.

Multiple Robots in Space: An Adaptive Eco-Grammar Model

Peter Sebestyén

Mathematical Institute, Silesian University,
Opava, Czech Republic,
Peter.Sebestyen@math.slu.cz

Petr Sosík

Institute of Computer Science, Silesian University,
Opava, Czech Republic,
and
Department of Computer Science, The University of Western Ontario,
London, ON, Canada, N6A 5B7
sosik@csd.uwo.ca

Abstract

We discuss aspects of multi-robot models suitable for study of interactions and emergence of rational behavior. To demonstrate advantages of the grammatical approach, we design an eco-grammatical model of adaptive multi-robot community. We demonstrate that this grammatical model can naturally involve a reinforcement collective learning. We test two learning algorithms in an environment with minimal communication of (almost) reactive robots. Experimental results show that using the eco-grammatical model, the robot community can be successfully trained to find a close-to-optimal solution to a given NP-complete task of a truss construction.

1 Models of Social Robot Behavior

The design and understanding of adaptive multi-agent systems is one of the most important challenges for theoretical robotics and artificial intelligence as well. There has been a movement in the last decade from the traditional idea of monolithic fabricated intelligence to the paradigm of *social intelligence*, see [1, 3] and others. An intelligent behavior and learning tends to be interpreted as a result of interaction of multiple autonomous agents. Examples of successful multi-agent robotic design include the walking six-legged robot Genghis [2], robotic soccer [10] or rescue robot team simulation [18], to name a few. In this paper we demonstrate that *eco-grammar* systems can be particularly useful for modelling and understanding emergence of social behavior of such simple agents.

There have been already reported some symbol- and grammar-based robotic models. For instance, Flann et al. [11] used a grammar-based approach to generate

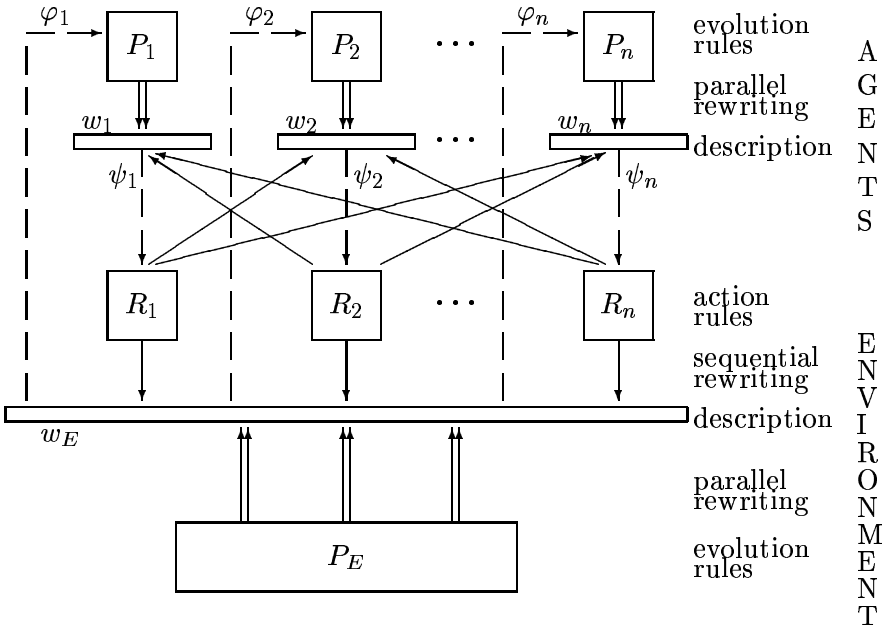

Figure 1: A basic scheme of eco-grammar system.

strategies for autonomous vehicles. Takadama et al. [17] studied a grammar-based robot community learning to perform a collective task. Dassow [9] used an eco-grammar system for modelling the behavior of the MIT robot Herbert. Due to [17], advantages of grammatical approach include (i) guarantee of a necessary level of rational behavior of each robot (ii) easy adaptation of robots to possible defective or inoperative functions (iii) autonomous behavior and emergence of rationality with minimal communication.

We add that (iv) clear interpretation of a robot's behavior and decision is obtained, (v) a successful implementation of collective learning algorithms is possible, and (vi) an analytical solution to the problem of deadlocks or cyclic behavior of the community is available under some restrictions.

2 Eco-Grammar Systems

The *eco-grammar* system was originally created as a grammatical model of interactions between an environment and organisms living in it. For motivation, examples and basic results of eco-grammar systems we refer to [5], [6].

Various models of eco-grammar systems have been studied, taking into the account direct mutual interactions between agents (predator – prey relations), prescribed teams of agents, prescribed scenarios (tables) of agent behaviors, multiplication and dying of agents etc. For these and further variants of eco-grammar systems we refer the reader to [7, 13, 14].

Prior to formal definitions, we fix some basic notation. For an alphabet V, we denote by V^+ the set of all nonempty strings over V. If the empty string, λ,

is included, then we use notation V^*. The length of a string x is denoted by $|x|$; $\mathrm{alph}(x) \subseteq V$ denotes the set of all letters contained in x. An *0L scheme* (an interactionless Lindenmayer scheme) is a pair $G = (V, P)$, where V is an alphabet and P is a complete set of context-free rewriting rules applied in parallel. For a set of context-free rules P we denote by $\mathrm{dom}(P)$ the set of left-hand sides of the rules in P, i.e. $\mathrm{dom}(P) = \{a \mid (a \to v) \in P\}$. We refer to [15] for further elements of formal language theory.

Definition 2.1 An *eco-grammar system* of degree $n, n \geq 1$, is an $(n+1)$-tuple $\Sigma = (E, A_1, \ldots, A_n)$, where

- $E = (V_E, P_E)$,
  - V_E is a finite alphabet and
  - P_E is a finite set of 0L rewriting rules over V_E, and
- $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i)$ for i, $1 \leq i \leq n$, where
  - V_i is a finite alphabet,
  - P_i is a finite set of 0L rewriting rules over V_i,
  - R_i is a finite set of rewriting rules of the form $x \to y$ with $x \in V_E^+$, $y \in V_E^*$,
  - $\varphi_i : V_E^* \longrightarrow 2^{P_i}$,
  - $\psi_i : V_i^+ \longrightarrow 2^{R_i}$.

(φ_i and ψ_i, $1 \leq i \leq n$, are always supposed to be computable functions but $\varphi_i(x), \psi_i(y)$ are not necessarily complete; 2^X denotes the power set of X.)

The above items are interpreted as follows:

- E represents the environment: V_E is the alphabet and P_E is the set of evolution rules.
- $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i)$ corresponds to an agent (the i-th agent), $1 \leq i \leq n$: V_i is the agent's alphabet, P_i is the set of evolution rules, R_i is the set of action rules. The mapping φ_i, depending on the state of the environment, selects the actual evolution rules. The mapping ψ_i selects the rules for the actual action depending on the current state of the agent.

Thus the 0L schemes (V_E, P_E) and (V_i, P_i), $1 \leq i \leq n$, describe the evolution of the environment and the agents, respectively. The production sets $R_1, \ldots, R_n$ define possible actions of the agents $A_1, \ldots, A_n$ within the environment. Figure 1 gives a schematic description of the system.

Definition 2.2 A *configuration* of an eco-grammar system Σ is a $(n+1)$-tuple

$$\sigma = (w_E; w_1, w_2, \ldots, w_n),$$

where $w_E \in V_E^*$ and $w_i \in V_i^*$, $1 \leq i \leq n$; w_E is called the *state* (evolution stage) of the environment, and w_i is the *state* of the i-th agent, $1 \leq i \leq n$.

In this paper we deal with the *weak* variant of eco-grammar system [4]. An agent A_i is said to be *active* in a configuration $(w_E; w_1, w_2, \ldots, w_n)$ if the set of its actual action rules $\psi_i(w_i)$ is nonempty. Agents can be active only if they are in a nonempty state; formally, $\psi_i(\lambda) = \emptyset$, $1 \leq i \leq n$.

A derivation step of an eco-grammar system from a configuration $(w_E; w_1, \ldots, w_n)$ to $(w'_E; w'_1, \ldots, w'_n)$ means:

(i) an application of a randomly chosen rule $r \in \psi_i(w_i)$ of each active agent A_i to a randomly chosen environment al symbol;

(ii) 0L rewriting of the environment w_E to w'_E by the productions of P_E in every place except the symbols already rewritten in phase (i);

(iii) a simultaneous 0L rewriting of every w_i to w'_i by productions of $\varphi_i(w_E)$.

If an agent A_i enters the empty state $w_i = \lambda$, then it remains inactive during the further functioning of the EG system. On the contrary, if $w_i \neq \lambda$ and no action rule of the agent is selected, i.e. $\psi_i(w_i) = \emptyset$, then the agent is only temporarily inactive. The same holds also if none of the agent's selected rules can be recently applied to the environment. For an example of functioning of an EG-system we refer to Section 3.1.

A uniform formal description of eco-grammar system allows one to solve analytically some important problems of behavior of a multi-robot community. For instance, consider the problem of detection of a *deadlock* (*blocking*): is there a situation during performing a common task when one or more robots block mutually their resources and cannot further act? Another problem is the one of *stagnating* behavior: are there situations when a team is trapped in a cycle? Under certain restrictions, answers to these questions can be effectively found in an eco-grammar system. The following result has been shown in [6].

Theorem 2.3 *There exists a polynomial-time algorithm (w.r.t. to the size of the system) deciding whether the development of a given unary deterministic simple EG system is blocking or stagnating.*

Also for *tabled* EG systems, in which agents switch between prescribed tables of behavior, a similar result has been shown [16]:

Theorem 2.4 *There exists an algorithm deciding for a conditional tabled $EG_1(i, 0; p)$ system, $i \geq 0$, whether there exists an infinite acyclic sequence of configurations (i.e. the system is non-stagnating).*

We refer the reader to [6, 8, 16] for further information on unary, deterministic and tabled eco-grammar systems.

3 A Case Study: Multiple Robots in Space

It follows by the previous description that the eco-grammar system's architecture corresponds to a team of behavior-based robots. There is a common environment

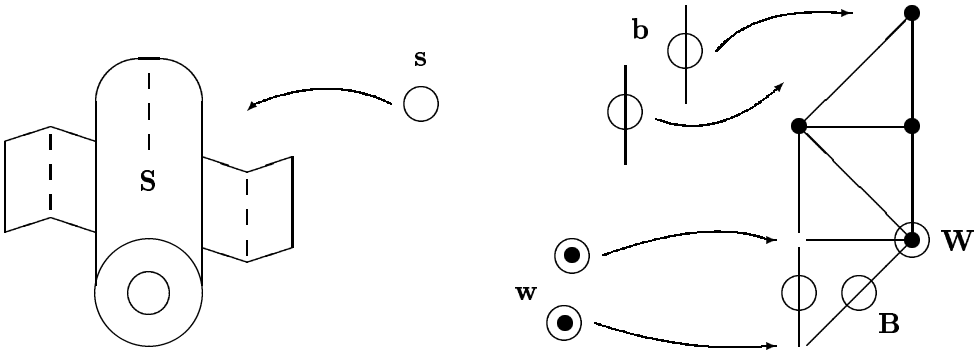

Figure 2: Multiple robots in space. Capital letters denote positions: **S** – space station, **W** – welding position, **B** – beam-holding position. Small letters denote actions: **b** – go to beam-holding position, **w** – go to welding position, **s** – go to station.

w_E, subject to an independent evolution rules P_E. A *perception* function for an agent A_i is defined by φ_i, an *action selection* function is ψ_i and action rules are R_i. The learning task for multiple robots described bellow has been originally presented in [17], where a hybrid model with grammatical aspects was presented.

Consider that there are m beams in a space station which have to be transported and weld together to construct a truss. There are n universal robots to participate at a solution. Each robot can perform one of the following actions:

- *transport a beam* from the station to the truss;
- *hold a beam* in a suitable position within the truss;
- *weld a beam* held by another robot;
- *transfer* between two locations.

These elementary actions can assemble to patterns of a robot behavior — *roles*, e.g. to transport a beam and then hold it until it is weld by another robot. The situation when the number of robots ready to weld is greater than the number of the other robots is considered as a *deadlock*. (The ready-to-weld robots block the access of the others to the truss.)

In our eco-grammar model the environment string w_E contains a global description of a current state of the task. The 0L scheme (V_E, P_E) defines the behavior and properties of the environment. Each robot is represented by an agent A_i, $1 \leq i \leq n$. Its internal state w_i represents its current position and a sequence of its previous actions for the purpose of training. Selection of a current role of the robot is driven by rules in the set P_i. Similarly as in [17], some rules in P_i have assigned individual weights that can be applied in a deterministic or probabilistic way.

Formally, consider a truss construction task with n robots and m beams. Let $\Sigma = (E, A_1, \ldots, A_n)$ be a weak EG system describing fully the task. Denote $d(X, Y)$ the number of steps to move from the position X to Y, for $X, Y \in \{S, B, W\}$.

- $V_E = \{F, G, \#\} \cup \{D_i, H_i, I_i, \#_i \mid 1 \le i \le n\} \cup \{D_{i[XY]} \mid 1 \le i \le n;\ X, Y \in \{B, S, W\}\}$;
- $P_E = \{D_{i[XY]} \to D_i^{d(X,Y)} \mid 1 \le i \le n,\ X, Y \in \{B, S, W\}\} \cup \{\#_i \to \# \mid 1 \le i \le n\}$ (recall that D^n denotes the string consisting of n symbols D) plus completing rules of the form $a \to a$ for the other symbols $a \in V_E$ not present on the left-hand side of the above rules.

Meaning of symbols in V_E is the following:

D_i	a travel distance element for the robot A_i;
G	a beam located in the station;
H_i	a beam transported by the robot A_i;
I_i	a beam held by the robot A_i in a beam holding position;
$\#$	a beam weld with the truss;
F	an auxiliary symbol – port for one robot.

Let $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i)$, $1 \le i \le n$, be a complete description of a robot.

- $V_i = \{S, B, W, W_x, P_S, P_B, P_W, s, b, w, R_{SB}, R_{BS}, R_{BW}, R_{WS}, R_{WW}, R_{SW}\}$;
- The mappings φ_i and ψ_i are defined as follows. Let for each rule $x \to y$ in P_i there be a set $C \subseteq V_E$ of *permitting* symbols and a set $D \subseteq V_E$ of *forbidding* symbols. Then

$$x \to y \in \varphi_i(w_E) \text{ iff } C \subseteq \mathrm{alph}(w_E) \text{ and } D \cap \mathrm{alph}(w_E) = \emptyset.$$

  The mapping ψ_i is defined in the same way w.r.t. the rules in R_i and the string w_i. Due to the above convention each rule in P_i or R_i will be denoted as $(C, D : x \to y)$. This type of regulation can be found in random context grammars [15] or conditional tabled eco-grammar systems [8].

- P_i contains the following rules:

$$\begin{array}{lll}
(\{G\}, \emptyset : S \to R_{SB})^\dagger & (\emptyset, \emptyset : R_{SB} \to P_B) & (\{D_i\}, \emptyset : P_W \to P_W) \\
(\{G\}, \emptyset : S \to R_{SW})^\dagger & (\emptyset, \emptyset : R_{BS} \to P_S) & (\{D_i\}, \emptyset : P_S \to P_S) \\
(\emptyset, \{H_i, I_i\} : B \to R_{BW})^\dagger & (\emptyset, \emptyset : R_{BW} \to P_W) & (\{D_i\}, \emptyset : P_B \to P_B) \\
(\emptyset, \{H_i, I_i\} : B \to R_{BS})^\dagger & (\emptyset, \emptyset : R_{WS} \to P_S) & (\emptyset, \{D_i\} : P_S \to sS) \\
(\{H_i\}, \emptyset : B \to B) & (\emptyset, \emptyset : R_{WW} \to P_W) & (\emptyset, \{H_i\} : P_B \to S) \\
(\{I_i\}, \emptyset : B \to B) & (\emptyset, \emptyset : R_{SW} \to P_W) & (\{H_i\}, \{D_i\} : P_B \to bB) \\
(\{\#_i\}, \emptyset : W_x \to wR_{WW})^\dagger & (\emptyset, \{\#_i\} : W_x \to W) & (\emptyset, \{D_i\} : P_W \to W) \\
(\{\#_i\}, \emptyset : W_x \to wR_{WS})^\dagger & (\{G\}, \emptyset : W \to W_x) &
\end{array}$$

$$(\emptyset, \{G\} \cup \{H_j, I_j \mid 1 \le j \le n\} : W \to R_{WS})$$
$$(\emptyset, \{G\} \cup \{H_j, I_j \mid 1 \le j \le n\} : S \to \lambda)$$

$$\left.\begin{array}{ll}
(\{H_j\}, \emptyset : S \to R_{SW})^\dagger & (\{H_j\}, \emptyset : W \to W_x) \\
(\{I_j\}, \emptyset : S \to R_{SW})^\dagger & (\{I_j\}, \emptyset : W \to W_x)
\end{array}\right\} 1 \le j \le n,$$

  plus completing rules of the form $a \to a$. Rules denoted by $\dagger$ can have assigned weights described in the next section.

- R_i contains the following action rules:

  $(\{R_{BS}\}, \emptyset : F \to FD_iD_{i[BS]})$ $(\{R_{WS}\}, \emptyset : F \to FD_iD_{i[WS]})$
  $(\{R_{BW}\}, \emptyset : F \to FD_iD_{i[BW]})$ $(\{R_{WW}\}, \emptyset : F \to FD_iD_{i[WW]})$
  $(\{R_{SW}\}, \emptyset : F \to FD_iD_{i[SW]})$ $(\{R_{SB}\}, \emptyset : G \to H_iD_iD_{i[SB]})$
  $(\emptyset, \emptyset : D_i \to \lambda)$ $(\{B\}, \emptyset : H_i \to I_i)$

  plus the set $\{(\{W\}, \emptyset : I_j \to \#_i) \mid 1 \leq j \leq n,\ j \neq i\}$.

Meaning of symbols in V_i, $1 \leq i \leq n$, is the following:

S	a robot is in the station;
B	a robot is in a beam holding location;
W, W_x	a robot is in a beam welding location;
P_Y	for $Y \in \{S, B, W\}$: a robot is in space on the way to location Y;
b	a robot transported a beam to a beam holding location;
s	a robot returned to the station;
w	a robot weld beams at a welding location;

3.1 An example of a simple task

As mentioned above, the rules in P_i denoted by † can have assigned weights. These rules determine the choice of current roles for robots. In our experiment the robots have two different main roles: *transport and hold a beam* and *(wait and) weld a beam*. For simplicity, we fix the distance between the space station and the truss to $d(S, B) = d(S, W) = 1$ time step and the distance between welding and beam-holding position within the truss to $d(B, W) = d(W, W) = 0$.

Example 3.1 Consider $n = 3$, let two robots A_1, A_2 transport and hold beams an let the third one A_3 weld beams. The weights will be set as follows:

	A_1	A_2	A_3
$S \to R_{SB}$	0.9	0.9	0.1
$S \to R_{SW}$	0.1	0.1	0.9
$B \to R_{BS}$	0.9	0.9	0.1
$B \to R_{BW}$	0.1	0.1	0.9
$W_x \to wR_{WS}$	0.9	0.9	0.1
$W_x \to wR_{WW}$	0.1	0.1	0.9

The sequence of configurations of the whole community follows, beginning in an initial configuration with all the beams and robots located in the station.

Step	Environment w_E	A_1	A_2	A_3
0.	F^3G^2	S	S	S
1.	F^3G^2	R_{SB}	R_{SB}	R_{SW}
2.	$F^3D_3D_{3[SW]}H_1D_1D_{1[SB]}H_2D_2D_{2[SB]}$	P_B	P_B	P_W
3.	$F^3D_3H_1D_1H_2D_2$	P_B	P_B	P_W
4.	$F^3H_1H_2$	P_B	P_B	P_W

5.	$F^3H_1H_2$	bB	bB	W
6.	$F^3I_1I_2$	bB	bB	W_x
7.	$F^3I_1I_2$	bB	bB	W

Now robots 1 and 2 are holding beams and robot 3 is ready to weld. After welding the robot 1's beam, robot 3 continues to another welding position, while robot 1 returns to the station. Then the same is done with the robot 2.

8.	$F^3\#_3I_2$	bB	bB	W_x
9.	$F^3\#I_2$	bR_{BS}	bB	wR_{WW}
10	$FFD_1D_{1[BS]}FD_3D_{3[WW]}\#I_2$	bP_s	bB	wP_W
11.	$FFD_1F\#I_2$	bP_s	bB	wP_W
12.	$F^3\#I_2$	bP_s	bB	wW
13.	$F^3\#\#_3$	bsS	bB	wW_x
14.	$F^3\#\#$	bs	bR_{BS}	wwR_{WW}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
23.	$F^3\#\#$	bs	bs	wws

In the final configuration all the beams are weld, all the robots are back at the station, state of each robot contains the history of its actions.

4 The Learning Algorithms

The task of coordination of multiple robots to fit into a given time limit is generally NP-complete. It is a case of so-called Precedence Constrained Multiprocessor Scheduling [12]. Hence, in larger robot communities, the problem of central control strategy is computationally unfeasible. The character of the task suggests as a solution the *Profit Sharing* (PS) reinforcement collective learning.

4.1 PS reinforcement learning

In the learning process the rules with assigned weights are selected by a roulette selection. After each task completion, positive or negative rewards are distributed to robots due to the result. Through the rewards distribution robots modify their own roles by changing the weight values of their individual rules. In accordance with [17], we modify the distribution of rewards to a *decreasing* (instead of original *increasing*) geometrical progression.

Positive rewards are distributed according to the formula:

$$W(r_i) := W(r_i) + R^+ \cdot (1/2)^{i-1}, \quad i = 1, ..., p. \tag{1}$$

Negative reward is distributed just to the last selected rule:

$$W(r_i) := W(r_i) - R^-, \text{ where } \; i = p. \tag{2}$$

In the above formulas $W(r_i)$ represent the weight of the rule r_i, R^+ and R^- are nonnegative reward values, p is the length of the current role-selection sequence of the robot. As an assumption, the weights of rules are set to the same value at the beginning of training.

4.2 Advanced reward assignment

As one can observe in Section 5, the robot community using the PS learning was able to learn the task successfully. If the ratio of beams/robots was high, however, the experimental results were worse than the estimated optimum. Furthermore, a large number of training cycles was necessary and a robustness of training was rather weak. For instance, in the case of 3 robots and 24 beams only one of 200.000 training cycles leaded to performing the task successfully.

We deduce that the PS learning algorithm did not motivate robots to effectivity, it just leaded to any non-deadlock solution. A improvement can be gained by scaling the positive reward value in dependence on the number of steps needed to complete the task:

$$W(r_i) := W(r_i) + (\ell/k)^{\alpha} \cdot R^{+} \cdot (1/2)^{i-1}, \quad i = 1, ..., p \ , \tag{3}$$

where ℓ is the estimated optimal length of the task, k is the number of steps in the success case and α is a positive integer. With growing value of α the reinforcement of weight values depends stronger on number of steps k. Moreover, unlike the previous section, the robots were trained to solve the task with subsequently increasing number of beams.

5 Simulation Results

We compare the number of steps needed for completing a given task in three experiments. In the first experiment used as a benchmark we perform the truss construction task described in section 3 with pre-set values of weights. This experiment called *Central Control.* In the second experiment named *PS Learning* we used the modified PS reinforcement learning from Section 4.1. In the third experiment denoted by *Advanced Learning* we applied the learning strategy from Section 4.2.

We performed all the experiments for the case of 13 and 24 beams. Figures 3 and 4 allows one to compare the results. When considering experimental results, we must remember that there is also certain number of "delay" steps due to the transfer of information in eco-grammar system between environment and the robots. An original Lisp implementation of eco-grammar system was used for the experiments.

5.1 Experiment with central control

In this experiment we set the weight values and hence divided the roles manually. Two main roles are denoted as follows:

- *HB* for transport and hold beam;
- *WB* for weld beam.

As the problem is NP-complete, we did not calculate an exact analytical minimum $s_{\min}(m, n)$ of steps depending on n (the number of robots) and m (the number of beams), for a given distances $d(X, Y)$, $X, Y \in \{B, S, W\}$. We can nevertheless give a certain upper and lower bound of this minimum.

For a lower bound, consider the case when the welding procedure and the distance $d(W, W)$ is very short, majority of the robots bring and hold beams and only a small

fraction of robots weld beams. Then clearly at least $\lfloor m/n \rfloor$ beam-holding cycles are necessary and hence

$$\left\lfloor \frac{m}{n} \right\rfloor \cdot c_1 + c_2 \leq s_{\min}(m,n) \tag{4}$$

for some constants $c_1 \geq 1$, and $c_2 \geq 0$ depending on $d(X,Y)$.

For an upper bound, assume on the contrary that the welding procedure and/or the transfer between welding locations takes more time than the beam bringing and holding. If we divide the robots to two teams of roughly the same size, the first one with the roles *HB* and the second one with the roles *WB*, we can certainly perform the task in $\lceil m/(2n) \rceil$ welding cycles, and hence

$$s_{\min}(m,n) \leq \left\lceil \frac{m}{2n} \right\rceil \cdot c_3 + c_4 \tag{5}$$

for positive constants c_3 and c_4. It follows from (4) and (5) that

$$s_{\min}(m,n) = \Theta\left(\frac{m}{n}\right). \tag{6}$$

In the experiment, robots performing *HB* had the weight values set as the robot A_1 in example 3.1. The *WB*-robots had the weight values as the robot A_3. We studied the cases from 3 to 15 robots and distributed the roles in a following way to estimate the optimal solution:

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
HB	*HB*	*WB*	*HB*	*WB*	*HB*	*HB*	*WB*	*HB*	*HB*	*WB*	*HB*	*HB*	*WB*	*HB*

5.2 Experiments with PS learning

The experiments with PS learning were performed for 3 to 15 robots. We used from 2,000 to 200,000 training cycles, reward values were $R^+ = 1$ and $R^- = 0.01$. The initial value of rule weights was set to 50. Since results of the experiments were nondeterministic due to the probabilistic learning algorithm, we have chosen for each value of n (the number of robots) the *best* result among 10 randomly chosen experiments.

5.3 Experiment with advanced learning

The robots were trained by a learning procedure described in Section 4.2. Denoting n the number of robots, we increased the number of beams by $2 * \lceil n/2 \rceil$ in each learning segment. This leaded to approx. ten times higher effectivity of learning — no more than 20.000 training cycles were necessary. Similarly as in the experiment with PS learning, the *best* result among 5 randomly chosen experiments was chosen. Unlike PS learning, this time more than 50% of the experiments resulted in the same value shown in the graph and only in a few cases the result was worse. Hence the advanced learning algorithm is much more robust. Moreover, the results were better for a higher ration m/n.

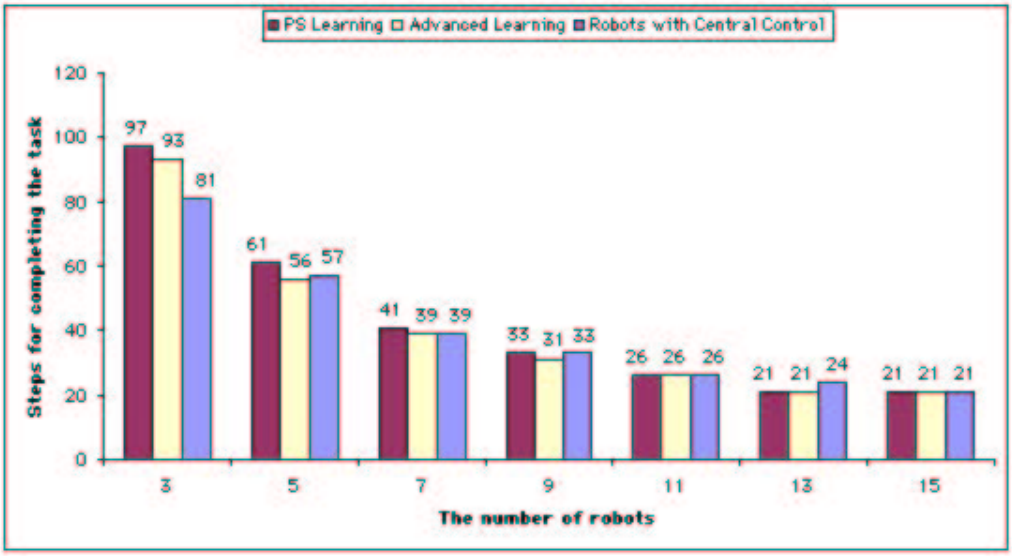

Figure 3: The experimental results for 13 beams

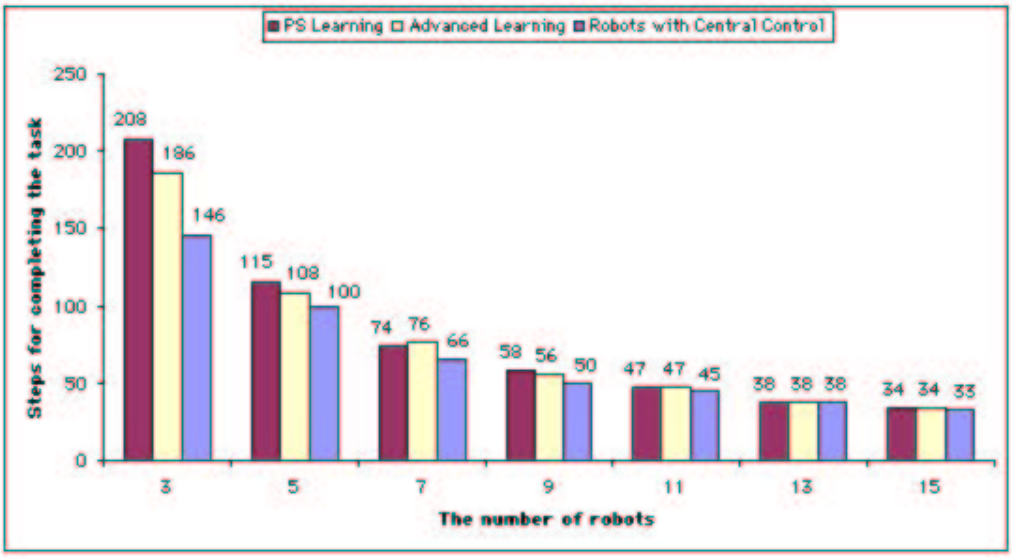

Figure 4: The experimental results for 24 beams

6 Discussion

We considered aspects of multi-agent and multi-robot modelling and discussed advantages of the grammatical approach. Clearly, this approach is appropriate in situations when details of physical interactions of robots and their environment are involved (e.g. robot vision, image recognition, control of drives etc.). But it provides a favorable tool on a certain level of abstraction when the environment, perception and actions of robots can be described by discrete means. We argue that substantial aspects of emergence of rational behavior and social intelligence could (and should)

be studied on such a level.

As an example we presented an eco-grammar model of and adaptive robot community in space. We tested in this framework two algorithms of distributed reinforcement learning: a modified PS learning and our advanced learning algorithm. The community of robots was able to perform the given task of the truss construction with both algorithms. Our advanced learning turned out to be more robust and also about ten times more effective. Rather surprisingly, the learning model has reached in many cases the estimated optimal solution.

Further theoretical and experimental study is necessary to exploit potential of grammatical models of robotic systems. Among many *open problems* we select a few:

- to study systematically models in which an analytical prediction of (un)desired configurations as a deadlock or cycles is possible;
- to accommodate the model with two- or three-dimensional environment, for the cases when the studied experiment would require to involve spatial interactions and topology;
- to study and model possible emergence of rational behavior in collectives of nanomechanisms as molecular motors, bio-arrays etc.

Acknowledgements

Research was partially supported by the Grant Agency of Czech Republic, Grant 201/02/P079 to P.S.

References

[1] R. M. Axelrod: *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration.* Princeton University Press, 1997.

[2] R. A. Brooks: A layered intelligent control system for a mobile robot. *IEEE Journal of Robotics and Automation* **2(1)** (1986), 14-23.

[3] R. A. Brooks: *Cambrian intelligence: the early history of the new AI.* MIT Press, Cambridge, Mass., 1999.

[4] J. Csima, Two remarks on variants of simple eco-grammar systems. *Acta Cybernetica* **14(4)** (2000), 569–582.

[5] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun: Eco-grammar systems. A preview. In R. Trappl (ed.): *Proc.* 12th *European Meeting on Cybernetics and System Research,* World Scientific, Singapore, 1994, 941–948.

[6] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun: Eco-grammar systems. A grammatical approach to life-like interactions. *Artificial Life* **3** (1997), 1–28.

[7] E. Csuhaj-Varjú, A. Kelemenová: Team behaviour in eco-grammar systems. *Theoretical computer science* **209** (1998), 213–224.

[8] E. Csuhaj-Varjú, Gh. Păun, A. Salomaa: Conditional tabled eco-grammar systems versus (E)T0L systems. *JUCS* **1(5)** (1995), 252–268.

[9] J. Dassow, An example of an eco-grammar system: a can collecting robot. In [13], 240–244.

[10] A. Drogoul, A. Collinot: Applying an agent-oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Autonomous Agents and Multi-Agent Systems* **1(1)** (1998), 113–129.

[11] N. S. Flann, M. Davidson, J. Martin, K. L. Moore: Intelligent behavior generation strategy for autonomous vehicles using a grammar-based approach. *International Conference on Field and Service Robotics*, Helsinki, University of Technology, Otaneimi, Espoo, Finland, 2001.

[12] M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman and Co., San Francisco, 1979.

[13] G. Păun (ed.): *Artificial Life: Grammatical Models.* The Black Sea University Press, Bucharest, 1995.

[14] G. Păun, A. Salomaa (eds.): *Grammatical models of multi-agent systems*, Gordon and Breach, Yverdon, 1999.

[15] A. Salomaa, G. Rozenberg (eds.): *Handbook of Formal Languages.* 3 volumes. Springer-Verlag, Berlin, 1997.

[16] P. Sosík: On the decidability problems of eco-grammar systems. *Journal of Automata, Languages and Combinatorics* **5** (45–58), 2000.

[17] K. Takadama, K. Hajiri, T. Nomura, K. Shimohara, S. Nakasuka: Grammatical learning model for adaptive collective behaviors in multiple robots. In G. Păun and A. Salomaa (Eds.): *Grammatical Models of Multi-agent Systems,* Gordon and Breach, London, 1999, 343–355.

[18] T. Takahashi, S. Tadokoro, M. Ohta, N. Ito: Agent Based Approach in Disaster Rescue Simulation - From Test-Bed of Multiagent System to Practical Application. RoboCup 2001, 102–111.

Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 313 - 327, 2004.

On metalinear CD grammar systems

Bettina Sunckel

Institut für Informatik Johann Wolfgang Goethe-Universität
D-60054 Frankfurt am Main, Germany
sunckel@psc.informatik.uni-frankfurt.de

Abstract

Metalinear CD grammar systems are defined to be context-free CD grammar systems where each component consists of metalinear productions. The maximal number of nonterminals in a starting production is the width of a CD grammar system. It will be shown that the width of metalinear CD grammar systems induces an infinite hierarchy of language classes. In addition it is established that metalinear CD grammar systems of a certain width generate language classes that do not contain all context-free languages but contain some context-sensitive languages. The resulting language classes are closed under union, intersection with regular languages, homomorphism and inverse homomorphism. They are not closed under concatenation, Kleene closure, intersection and complement.

1 Introduction

There are many ways to combine the classical formal language theory with the concept of distribution. Grammar systems are a combination of formal grammars and distribution. A grammar systems consist of several components where each component is a grammar. Cooperating distributed (CD) grammar systems are a sequential model where at one point of time only one component (the active component) contributes to the derivation. A good overview of the topic can be found in [2] or in [4] and also a connection to artificial intelligence is given in [2].

From the language theoretic point of view it is desirable to find grammar models that combine the simplicity of context-free rules with the power of generating some context-sensitive languages. Context-free CD grammar systems consist of several context-free grammars and can be seen as a generalization of context-free grammars. The derivation mode defines how long productions of one component can be used and when the next component becomes active. Apart from the t-mode where each component contributes to the derivation as long as possible, we will consider the $(= k)$- and $(\geq k)$-mode where each component performs exactly k or $\geq k$ steps, respectively. In case of the t-mode the class of $ET0L$ languages is generated and using the $(= k)$ and$(\geq k)$-mode we obtain a subclass of the language class generated by matrix grammars [2]. Thus, the concept of distribution adds generative power to context-free productions.

However, context-free CD grammar systems are not as easy to handle as context-free grammars. For instance, it is not known whether a tool like a pumping lemma exists. Hence, it is difficult to exclude a certain language from a certain language class generated by CD grammar systems. It is therefore natural to find subclasses which are easier to use but still contain interesting non context-free languages. When considering context-free languages, the most simple form of a production is a linear production. But we will show that CD grammar systems with linear productions only generate linear context-free languages. Metalinear context-free grammars are only allowed to have more than one nonterminal in a starting production. In this way the derivation trees stay nearly as simple as linear derivation trees.

We define metalinear CD grammar systems to be CD grammar systems with metalinear context-free grammars as components and prove a pumping lemma similar to that for metalinear context-free languages. Then it is established that the width, which is the maximal number of nonterminals in a starting production, induces an infinite hierarchy of language classes. Also some interesting context-sensitive languages that can be generated by metalinear CD grammar systems are presented. However, it is not possible to generate all context-free languages. Furthermore, some nice closure properties are proven such as closure under homomorphism, inverse homomorphism, union and intersection with regular sets. The resulting language classes are not closed under under concatenation, Kleene closure intersection and complement.

2 Definitions

The basics of formal language theory can be found in [5]. Let A be a finite alphabet. We denote the empty string by ε and the Kleene closure of A by A^*. Let B be a finite set. With $|B|$ we refer to the cardinality of B and with B' we refer to $\{a' \mid a \in B\}$.

A context-free grammar $G = (N, T, P, S)$ is linear, if each rule is of the form $A \to uBv$ or $A \to u$ where $A, B \in N$ and $u, v \in T^*$. A language L is said to be linear if there exists a linear grammar G such that $L(G) = L$.

A context-free grammar $G = (N, T, P, S)$ is m-linear, if each rule is of the form $S \to A_1 \dots A_{m_0}$, $A \to uBv$ or $A \to u$ where $A, A_1, \dots, A_{m_0}, B \in (N \setminus \{S\})$, $m_0 \leq m$ and $u, v \in T^*$. A language L is m-linear, if there is an m-linear grammar G such that $L(G) = L$. If a grammar or language is m-linear for some $m \geq 2$, it is also called metalinear.

We denote the class of regular languages by REG, the class of context-free languages by CF and the class of linear languages by LIN. For the class of m-linear languages we will write $mLIN$ and $METALIN := \bigcup_{i \geq 1} iLIN$.

Definition 2.1. A CD grammar system is a $k+3$ tuple $\Gamma = (N, T, P_1, \dots, P_k, S)$, where N is a finite set of nonterminals, T is a finite set of terminals, P_i, $1 \leq i \leq k$ is a finite set of rules and $S \in N$ is the axiom.

In the following, we will write $\beta_1 X \beta_2 \Rightarrow_{P_i} \beta_1 \alpha \beta_2$ if $X \to \alpha$ is in P_i and β_1, β_2 are in $(N \cup T)^*$. We say that the production $X \to \alpha$ is applied to the derivation string $\beta_1 X \beta_2$. We denote reflexive and transitive closure of $\Rightarrow_{P_i}$ by $\Rightarrow^*_{P_i}$ and we write

$\alpha \Rightarrow_{P_i}^{k} \beta$, if α derives β in exactly k steps. We define the domain of a component as $Dom(P_i) := \{A \mid A \to \alpha \in P_i\}$.

Definition 2.2. Let Γ be a CD grammar system.

1. For each i, $1 \leq i \leq n$, a terminating derivation in the i-th component is $x \Rightarrow_{P_i}^{t} y$ iff $x \Rightarrow_{P_i}^{*} y$ and there is no $z \in (N \cup T)^*$ with $y \Rightarrow_{P_i} z$.

2. For each i, $1 \leq i \leq n$, a k-steps derivation in the i-th component is $x \Rightarrow_{P_i}^{=k} y$ iff there are $x_1, ..., x_{k+1} \in (N \cup T)^*$ such that $x = x_1, y = x_{k+1}$ and $x_j \Rightarrow_{P_i} x_{j+1}, 1 \leq j \leq k$.

3. For each i, $1 \leq i \leq n$, an at most k-steps derivation in the i-th component is $x \Rightarrow_{P_i}^{\leq k} y$ iff $x \Rightarrow_{P_i}^{=k'} y$ for some $k' \leq k$.

4. For each i, $1 \leq i \leq n$, an at least k-steps derivation in the i-th component is $x \Rightarrow_{P_i}^{\geq k} y$ iff $x \Rightarrow_{P_i}^{=k'} y$ for some $k' \geq k$.

Now we define metalinear CD grammar systems as CD grammar systems that consist of context-free metalinear grammars as components.

Definition 2.3. A CD grammar system is called m-linear for a fixed $m \geq 2$, if each production is formed as follows: $S \to A_1 \dots A_{m_0}$, $A \to uBv$, $A \to u$ with $A, B, A_1, \dots, A_{m_0} \in (N \setminus \{S\})$, $m_0 \geq m$ and $u, v \in T^*$. If a CD grammar system is m-linear for some $m \geq 2$, it is also called metalinear. We refer to m as the width of an m-linear CD grammar system.

We denote the class of m-linear CD grammar systems with l components by CD_l-*mLIN* and the class of metalinear CD grammar systems with l components by CD_l-*METALIN*. For the class of languages which are generated by grammars in CD_l-*mLIN* or CD_l-*METALIN* in derivation mode $f \in \{*, t, = k, \geq k, \leq k \mid k \geq 1\}$ we write $\mathcal{L}_f(CD_l\text{-}mLIN)$ or $\mathcal{L}_f(CD_l\text{-}METALIN)$, respectively. We omit the number of components, if no restrictions are made.

Definition 2.4. An $ET0L$ system is a (n+3)-tuple $G = (V, T, P_1, \dots, P_n, w)$ where V is the total alphabet, $T \subseteq V$ is the terminal alphabet, $P_i, 1 \leq i \leq n$ are the tables consisting of context-free rules, $w \in V^*$ is the start string.

Each table is complete, that is for all symbols $a \in V$ it contains at least one rule of the form $a \to x$. In the following rules of the form $a \to a$ are not explicitly written. A symbol a in an $ET0L$ system is called active if there is a table in G with a production $a \to x$ where $a \neq x$. The number of active symbols in the string $\alpha \in V$ is denoted by $\#_{A(G)}(\alpha)$.

Definition 2.5. An $ET0L$ system $G = (V, T, P_1, \dots, P_n, S)$ is called m-linear if and only if it has the following properties. $S \in V \setminus T$ does not appear at the right hand side of any production. If $S \Rightarrow_G \alpha$ then $\#_{A(G)}(\alpha) \leq k$ Every production whose left-hand side is not S is linear. An $ET0L$ system is called metalinear if and only if it is m-linear for some m.

Metalinear $ET0L$ systems are investigated in [6]. We will denote the class of m-linear ET0L systems with $ET0L_{mLIN}$, the class of metalinear ET0L systems with $ET0L_{METALIN}$ and the corresponding language classes with $\mathcal{L}(ET0L_{mLIN})$ and $\mathcal{L}(ET0L_{METALIN})$.

Definition 2.6. A context-free matrix grammar is a 4-Tuple $G = (N, T, M, S)$ where N is a set of nonterminals, T is a set of terminals, M a finite set of sequences $s : (r_1, r_2, \ldots, r_{n_s}), n_s \geq 1$ with $r_i \in N \times (T \cup N)^*$ and S is the axiom.

A derivation step of a matrix grammar consists of the sequential application of the rules $r_1, \ldots, r_{n_m}$ to the derivation string.

Definition 2.7. A matrix grammar $G = (N, T, M, S)$ is called m-linear if and only if each production is formed as follows: $S \to A_1 \ldots A_{m_0}$, $A \to uBv$, $A \to u$ with $A, B, A_1, \ldots, A_{m_0} \in (N \setminus \{S\})$, $m_0 \geq m$ and $u, v \in T^*$. An matrix grammar is called metalinear if and only if it is m-linear for some m.

Definition 2.8. Let Γ be a CD grammar system. A tree is a derivation tree of Γ iff the following holds. Every vertex has a label which is a symbol of $N \cup T \cup \{\varepsilon\}$. The label of the root is S. If a vertex is interior and has label A, then A must be in N. If n has label A and vertices $n_1, n_2, \ldots, n_k$ are sons of vertex n, in order from the left, with labels $X_1, X_2, \ldots X_k$, respectively, then $A \to X_1 X_2 \ldots X_k$ must occur in one of the components of Γ. If vertex n has label ε, then n is a leaf and is the only son of its father.

The labels of the leaves of a derivation tree read from left to right are called the *yield* of the tree. A *subtree* of a derivation tree is a particular vertex together with all its descendants. It should be noted that each derivation of a CD grammar system corresponds to a derivation tree, but there are also derivation trees which do not correspond to a derivation of the underlying CD grammar system. We call derivation trees which correspond to a derivation of Γ *valid*. If a valid derivation tree has a yield $w \in T^*$ we call it a *valid complete* derivation tree.

3 Examples

In this section we will show, that there are interesting non context-free languages which can be generated by m-linear CD grammar systems. Furthermore there are m-linear context-free languages which can be generated by $(m-1)$-linear CD grammar systems.

Example 3.1. Consider the language $L_1 = \{a_1^n a_2^n b_1^m b_2^m c_1^l c_2^l \mid l, m, n \geq 1\}$. The following 2-linear grammar system will generate L_1 using the t-mode. L_1 is known to be 3-linear context-free.

$$\begin{aligned}
\Gamma_1 &= \{\{S, A, C, B_1, B_2, B_1', B_2'\}, \{a, b, c\}, P_1, P_2, P_3, P_4, S\}, \\
P_1 &= \{S \to B_1 B_2, B_1 \to B_1' b_1, B_2 \to b_2 B_2'\}, \\
P_2 &= \{B_1' \to B_1, B_2' \to B_2\}, \\
P_3 &= \{B_1 \to A b_1, B_2 \to b_2 C\}, \\
P_4 &= \{A \to a_1 A a_2, C \to c_1 C c_2, A \to a_1 a_2, C \to c_1 c_2\}.
\end{aligned}$$

With the component P_1 the two nonterminals B_1 and B_2 are inserted. After that, the numbers of b_1s and b_2s will be increased with the components P_1 and P_2. The derivation of bs is finished with component P_3. Component P_4 inserts as and cs like a context-free grammar and completes the derivation.
$S \Rightarrow_{P_1} B_1'b_1b_2B_2' \Rightarrow_{P_2} B_1b_1b_2B_2 \Rightarrow_{P_1} \ldots \Rightarrow_{P_2} B_1b_1^{m-1}b_2^{m-1}B_2 \Rightarrow_{P_3} Ab_1^m b_2^m C \Rightarrow_{P_4} a_1^n a_2^n b_1^m b_2^m c_1^l c_2^l$.

Example 3.2. The language $L_2 = \{ww \mid w \in \{a,b\}^+\}$ is generated by the following grammar system using the $(=2)$-mode:

$$\begin{aligned}
\Gamma_2 &= \{\{S, S', A, B, A', B'\}, \{a, b\}, P_1, P_2, P_3, S\}, \\
P_1 &= \{S \to S', S' \to AB, A \to aA', B \to aB', A \to a, B \to a\}, \\
P_2 &= \{A \to bA', B \to bB', A \to b, B \to b\}, \\
P_3 &= \{A' \to A, B' \to B\}.
\end{aligned}$$

The following derivation shows how Γ_2 works.
$S \Rightarrow_{P_1} AB \Rightarrow_{P_1} aA'aB' \Rightarrow_{P_3} aAaB \Rightarrow_{P_2} abA'abB' \Rightarrow_{P_3} abAabB \Rightarrow_{P_1} abaaba$

Example 3.3. The language $L_3 = \{a^n b^n a^n b^n a^n b^n \mid n \geq 0\}$ is generated by the following grammar system using the $(=2)$-mode

$$\begin{aligned}
\Gamma_3 &= \{\{S, A, B, C, A', B', C', B'', B''', B''''\}, \{a, b\} \\
&\quad , P_1, P_2, P_3, P_4, P_5, P_6, P_7, S\} \\
P_1 &= \{S \to S', S' \to ABC\}, \\
P_2 &= \{A \to aA'b, B \to aB'b\}, \\
P_3 &= \{B' \to B'', C \to aC'b\}, \\
P_4 &= \{A' \to A, B'' \to B'''\}, \\
P_5 &= \{B''' \to B, C' \to C\}, \\
P_6 &= \{A \to \varepsilon, B \to B''''\}, \\
P_7 &= \{B'''' \to \varepsilon, C \to \varepsilon\}.
\end{aligned}$$

The derivations of Γ_3 are formed as follows:
$S \Rightarrow_{P_1} ABC \Rightarrow_{P_2} aA'baB'bC \Rightarrow_{P_3} aA'baB''baC'b \Rightarrow_{P_4}$
$aAbaB'''baC'b \Rightarrow_{P_5} aAbaBbaCb(\Rightarrow_{P_2} \ldots \Rightarrow_{P_3} \ldots \Rightarrow_{P_4} \ldots \Rightarrow_{P_5})^{n-1}$
$a^n Ab^n a^n Bb^n a^n Cb^n \Rightarrow_{P_6} a^n b^n a^n B''''b^n a^n Cb^n \Rightarrow_{P_7} a^n b^n a^n b^n a^n b^n$

4 Generative Capacity

4.1 Pumping Lemma

Considering the generative capacity of languages we need the possibility to decide, if certain languages are included in a language class. For metalinear CD grammar systems it will be possible to obtain a pumping lemma. This will help us to establish a hierarchy of language classes consisting of languages generated by metalinear CD grammar systems.

Lemma 4.1. *Let $f \in \{t, = k, \geq k \mid k \geq 2\}$ and $L \in \mathcal{L}_f(CD_l\text{-}mLIN)$ be an infinite language. Then there is a $h \in \mathbb{N}$ and for all w with $|w| \geq h$ such that $w = c_1 \dots c_m$ and*

1.
   - $c_i = x_i u_i y_i v_i z_i$, $1 \leq i \leq m$,
   - $u_1 v_1 u_2 v_2 \dots u_m v_m \neq \varepsilon$,
   - *for all i_0, $1 \leq i_0 \leq m$ with $|c_{i_0}| > h/m$ it is $u_{i_0} v_{i_0} \neq \varepsilon$ and $|x_{i_0} u_{i_0} v_{i_0} z_{i_0}| \leq h/m$,*
   - *for all $j, j \geq 0$ $x_1 u_1^j y_1 v_1^j z_1 x_2 u_2^j y_2 v_2^j z_2 \dots x_m u_m^j y_m v_m^j z_m \in L$.*
2. *for all i_0, $1 \leq i_0 \leq m$ with $|c_{i_0}| > h/m + s$ and each substring c'_{i_0} with $|c'_{i_0}| > h/m + s$ there is a γ with*
   - *$c'_{i_0}\gamma$ is a substring of c_{i_0},*
   - *$c'_{i_0}\gamma = xuyvz$ and xu is a substring of the first $h/m + s$ terminals of c'_{i_0} and $|u| > 0$,*
   - *for all $j, j \geq 0$ $\phi x u^j y v^j z \phi' \in L$.*

   or

   - *$\gamma c'_{i_0}$ is a substring of c_{i_0},*
   - *$\gamma c'_{i_0} = xuyvz$ and vz is a substring of the last $h/m + s$ terminals of c'_{i_0} and $|u| > 0$,*
   - *for all $j, j \geq 0$ $\phi x u^j y v^j z \phi' \in L$.*

Proof. We will first prove the claim for the t-mode of derivation.
1. Let $L \in \mathcal{L}_t(CD_l\text{-}mLIN)$ an infinite language and $G \in CD_l\text{-}mLIN$ be a grammar system that generates L using the t-mode. Furthermore let s be the maximum number of terminals in one Production of G, p the maximum number of productions with terminals in one component and r the number of nonterminals in G. Now, consider a word $w \in L$ consisting of n terminals. Then there were at least n/s productions used in a derivation and the corresponding derivation tree of this word. Let $h = s \cdot (r \cdot l \cdot p + 1) \cdot m^2$ and $w \in L$ with $|w| \geq h$. Then w is divided in m parts $w = c_1 \dots c_m$ with $S \Rightarrow A_1 \dots A_m$ and $A_i \Rightarrow c_i$ and $1 \leq i \leq m$. There is at least one $c_{i_0} \geq s \cdot (r \cdot l \cdot p + 1)$. Now, consider the subtree with root A_{i_0} that yields c_{i_0}. Note that only linear productions are used in this subtree. We analyze the first $l \cdot r \cdot p + 1$ productions with terminals.
Case 1: When deriving c_{i_0} one of the components uses more than p productions that contain terminals. Then there are two vertices vt_1 and vt_2 satisfying the following conditions.

- Both vertices have the same label and the same productions applied to them.
- Vertex vt_1 is closer to the root than vertex vt_2.
- The potion of the path from vt_1 to vt_2 is of length at most p

In this case we have $x_h, u_h, v_h, z_h = \varepsilon,\ 1 \leq h \leq m,\ h \neq j$ and

$$\begin{aligned} w &= y_1 \dots x_j u_j y_j v_j z_j \dots y_m \\ &= x_1 u_1 y_1 v_1 z_1 \dots x_j u_j y_j v_j z_j \dots x_m u_m y_m v_m z_m. \end{aligned}$$

When the productions on the path from vt_1 to vt_2 are applied i times, we obtain the word

$$\begin{aligned} w' &= y_1 \dots x_j u_j^i y_j v_j^i z_j \dots y_m \\ &= x_1 u_1^i y_1 v_1^i z_1 \dots x_j u_j^i y_j v_j^i z_j \dots x_m u_m^i y_m v_m^i z_m. \end{aligned}$$

Furthermore $0 < |x_{j_0} u_{j_0} v_{j_0} z_{j_0}| \leq l \cdot p + 1 \leq h/m$.
Case 2: When deriving w none of the components uses more than p productions that contain terminals. Therefore there have to be at least $(r \cdot l + 1) \cdot m$ changes of components in the derivation.

Now, consider the leaves of the derivation tree of w when a component stops working. The leaves are a sequence of at least m nodes each labelled with a terminal or a nonterminal. Since terminals in the derivation tree can not be changed, they are represented as $\times$ in the sequence. Hence, there are only $(r + 1) \cdot m$ different sequences for nonterminals at the leaves of the derivation tree. One of the sequences has therefore to occur twice while deriving w. Let

$$(\times_1, \dots, \times_{j_1 - 1}, B_{j_1}^1, \times_{j_1+1}, \dots, \times_{j_p - 1}, B_{j_p}^1, \times_{j_p+1}, \dots, \times_m)$$

and

$$(\times_1, \dots, \times_{j_1 - 1}, B_{j_1}^2, \times_{j_1+1}, \dots, \times_{j_p - 1}, B_{j_p}^2, \times_{j_p+1}, \dots, \times_m)$$

be these sequences where each position stands for one of the m branches of the derivation tree and $\times$ means that there is a terminal at the end of this branch. In this case $x_h, u_h, v_h, z_h = \varepsilon,\ 1 \leq h \leq m,\ h \neq j_1 \dots j_p$. We have

$$\begin{aligned} w &= y_1 \dots y_{j_1 - 1} x_{j_1} u_{j_1} y_{j_1} v_{j_1} z_{j_1} y_{j_1+1} \dots y_{j_p - 1} x_{j_p} u_{j_p} y_{j_p} v_{j_p} z_{j_p} y_{j_p+1} \dots y_m \\ &= x_1 u_1 y_1 v_1 z_1 \dots x_{j_1 - 1} u_{j_1 - 1} y_{j_1 - 1} v_{j_1 - 1} z_{j_1 - 1} x_{j_1} u_{j_1} y_{j_1} v_{j_1} z_{j_1} \\ &\quad x_{j_1+1} u_{j_1+1} y_{j_1+1} v_{j_1+1} z_{j_1+1} \dots \\ &\quad x_{j_p - 1} u_{j_p - 1} y_{j_p - 1} v_{j_p - 1} z_{j_p - 1} x_{j_p} u_{j_p} y_{j_p} v_{j_p} z_{j_p} \\ &\quad x_{j_p+1} u_{j_p+1} y_{j_p+1} v_{j_p+1} z_{j_p+1} \dots x_m u_m y_m v_m z_m. \end{aligned}$$

If the productions used to change the tree from configuration

$$(\times_1, \dots, \times_{j_1 - 1}, B_{j_1}^1, \times_{j_1+1}, \dots, \times_{j_p - 1}, B_{j_p}^1, \times_{j_p+1}, \dots, \times_m)$$

to configuration

$$(\times_1, \dots, \times_{j_1 - 1}, B_{j_1}^2, \times_{j_1+1}, \dots, \times_{j_p - 1}, B_{j_p}^2, \times_{j_p+1}, \dots, \times_m)$$

are applied i times one gets the following word

$$\begin{aligned} w' &= y_1 \dots y_{j_1-1} x_{j_1} u^i_{j_1} y_{j_1} v^i_{j_1} z_{j_1} y_{j_1+1} \dots y_{j_p-1} x_{j_p} u^i_{j_p} y_{j_p} v^i_{j_p} z_{j_p} y_{j_p+1} \dots y_m \\ &= x_1 u^i_1 y_1 v^i_1 z_1 \dots x_{j_1-1} u^i_{j_1-1} y_{j_1-1} v^i_{j_1-1} z_{j_1-1} x_{j_1} u^i_{j_1} y_{j_1} v^i_{j_1} z_{j_1} \\ & \quad x_{j_1+1} u^i_{j_1+1} y_{j_1+1} v^i_{j_1+1} z_{j_1+1} \dots \\ & \quad x_{j_p-1} u^i_{j_p-1} y_{j_p-1} v^i_{j_p-1} z_{j_p-1} x_{j_p} u^i_{j_p} y_{j_p} v^i_{j_p} z_{j_p} \\ & \quad x_{j_p+1} u^i_{j_p+1} y_{j_p+1} v^i_{j_p+1} z_{j_p+1} \dots x_m u^i_m y_m v^i_m z_m \end{aligned}$$

with $w' \in L$. Furthermore $0 < |x_{j_0} u_{j_0} v_{j_0} z_{j_0}| \leq l \cdot p + 1 \leq h/m$.
2. As shown in 1. c_{i_0} is the yield of a linear subtree t of the derivation tree of w. When we complete c'_{i_0} with the appropriate γ we obtain the yield of a subtree t'' of t'. Since $|c'_{i_0}| > h/m$, t'' consists of at most $h/m/s$ productions with terminals. We will only consider the case $yield(t'') = c'_{i_0}\gamma$ because the other one can be proven analogously. We have shown in 1. that in a subtree consisting of $h/m/s$ productions either one component has applied more than p productions (case 1) or there have been at least $(r \cdot l + 1) \cdot m$ changes of components (case 2). In both cases there is a vertex vt_1 and a vertex vt_2 of t'' labelled with the same nonterminal. The path from vt_1 to vt_2 is not longer than p or $(r \cdot l + 1) \cdot m$, respectively. Hence $c'_{i_0}\gamma = xuyvz$. When the productions from vt_1 to vt_2 are applied j times we obtain xu^jyv^jz. Since $c'_{i_0} > h/m + s$ we have xu is a substring of the first $h/m + s$ terminals of c'_{i_0} and $|u| > 0$. We have shown in 1. that it is possible to apply the productions from vt_1 to vt_2 j times (possibly together with productions in other subtrees) and obtain a word in L. Therefore there are ϕ and ϕ' and for all $j, j \geq 0$ $\phi xu^jyv^jz\phi' \in L$.

Now we consider the $= k$ mode of derivation. Since only k productions are applied in one component the proof of 1. and 2. is similar to case 2 of the proof of Lemma 4.1. Case 1 is not applicable, because one component has to use exactly k components and pumping within these components is not possible.

The proof for the $\geq k$ mode is similar to the proof for the t mode since here also both cases of pumping can occur. □

4.2 Generative Capacity Results

With the above pumping lemma we first show, that there are context-free languages that can not be generated by metalinear CD grammar systems.

Theorem 4.1. *CF $\not\subset \mathcal{L}_f$(CD-mLIN) for all $m \geq 2$ and $f \in \{t, = k, \geq k \mid k \geq 2\}$.*

Proof. Consider the language $L = \{a^nb^n \mid n \geq 0\}^*$. This language is context-free. Now we assume that L is in $\mathcal{L}_f(CD_l\text{-}mLIN)$ for $f \in \{t, = k, \geq k \mid k \geq 2\}$ and an arbitrary $m \geq 2$. Consider the word $w = (a^hb^h)^{3m+1}$ where h is the constant from Lemma 4.1. Then $w = c_1 \dots c_m$. It follows that there is a c_{i_0} with $|c_{i_0}| > 6h$. According to Lemma 4.1 we choose a substring $c'_{i_0} = a^{h/m+s}b^ha^hb^{h/m+s}$ of c_{i_0}. Note that this is possible because of the length of c_{i_0}. Then c'_{i_0} is divided as follows $c'_{i_0} = xuyvz$.
Case 1: c'_{i_0} is continued to $c'_{i_0}\gamma$. Then $|xuvz| < h/m$, $|u| > 0$ and therefore only contains as from the first part of c'_{i_0}. It follows that $y > 4h$ and thus contains b^{2h} in the first half of c'_{i_0}. The word $w' = \phi xu^jyv^jz\phi'$ contains at least one pair $a^{2h+x}b^{2h}$

where the number of as is not equal to the number of bs. The word w' is not in L.
Case 2: c'_{i_0} is continued to $\gamma c'_{i_0}$. A word $w' = \phi x u^j y v^j z \phi'$ which contains at least one pair $a^{2h}b^{2h+x}$ where the number of as is not equal to the number of bs can be constructed similar as in case 1. The word w' is not in L.
It follows that L is not in $\mathcal{L}_f(CDmLIN)$ for $f \in \{t, = k, \geq k \mid k \geq 2\}$ and an arbitrary $m \geq 2$. □

Definition 4.1. For all $n \in \mathbb{N}$ let $L_n = \{(a^i b^i)^n \mid i \geq 1\}$.

Theorem 4.2. *Let $f \in \{t, = k, \geq k \mid k \geq 2\}$ and $n \geq 2$. The language L_n is in $\mathcal{L}_f$(CD-nLIN) but not in $\mathcal{L}_f$(CD-$(n-1)$LIN).*

Proof. The following grammar system in *CD-nLIN* can generate L_n in the t-mode: Let $G_n = (\{S, A, B\}, \{a, b\}, P_1 \dots P_5, S)$ and

$$\begin{aligned}
P_1 &= \{S \to A^n\},\\
P_2 &= \{A \to aBb\},\\
P_3 &= \{B \to aAb\},\\
P_4 &= \{A \to ab\},\\
P_{.5} &= \{B \to ab\}
\end{aligned}$$

The components P_2 or P_3 add an equal number of as and bs two the n substrings derived from the n nonterminals A which are added with the starting production. Component P_4 or P_5 terminate a derivation.
We will now give a grammar system in *CD-nLIN* that generates L_n in the $(= 2)$- or (≥ 2)-mode. Let $G_n = (\{S, B, A_1, A'_1, A''_1, A'''_1, A''''_1, \dots, A_n, A'_n, A''_n, A'''_n, A''''_n\}, \{a, b\}, P_1 \dots P_{3n-2}, S)$ and

$$\begin{aligned}
P_1 &= \{S \to BA_2 \dots A_n, B \to A_1\},\\
P_2 &= \{A_1 \to aA'_1b, A_2 \to A''_2\},\\
P_3 &= \{A''_2 \to aA'_2b, A_3 \to A''_3\},\\
&\dots\\
P_{n-1} &= \{A''_{n-2} \to aA'_{n-2}b, A_{n-1} \to A''_{n-1}\},\\
P_n &= \{A''_{n-1} \to aA'_{n-1}b, A_n \to aA'_nb\},\\
P_{n+1} &= \{A'_n \to A_n, A'_{n-1} \to A'''_{n-1}\},\\
P_{n+2} &= \{A'_{n-2} \to A'''_{n-2}, A'''_{n-1} \to A_{n-1}\},\\
&\dots\\
P_{2n-1} &= \{A'_1 \to A_1, A'''_2 \to A_2\},\\
P_{2n} &= \{A_1 \to ab, A_2 \to A''''_2\},\\
P_{2n+1} &= \{A''''_2 \to ab, A_3 \to A''''_3\},\\
&\dots\\
P_{3n-2} &= \{A''''_{n-1} \to ab, A_n \to ab\}.
\end{aligned}$$

The derivation begins with P_1. After that P_2 to P_n add one a and one b to the n substrings derived from A_1 to A_n. Note that these components can only be applied in the given order. The resulting derivation string contains only primed nonterminals. Then the components P_{n+1} to P_{2n-1} change the nonterminals in the derivation string to unprimed nonterminals. These components can also only be applied in the given order. The derivation is finished with the components P_{2n} to P_{3n-1} which can again be only applied in the given order.
Now we assume that L_n is in $\mathcal{L}_f$*(CD-*$(n-1)$*LIN)*. Let h be the constant known from the pumping lemma. We choose a word w in L_n with $|w| > h$. Then $w = (a^i b^i)^n$. But we know from Lemma 4.1 that

$$w = x_1 u_1 y_1 v_1 z_1 x_2 u_2 y_2 v_2 z_2 \dots x_n u_n y_n v_n z_n$$

and the following holds: At least one of the u_j or v_j is not the empty word. All u_j and v_j only consist of as only consist of bs or else we can obtain a word not in L_n. There are at most $2(n-1)$ u_j and v_j. But if at least one and at most $2(n-1)$ of the substrings a^n or b^n are pumped to $a^{n'}$ or $b^{n'}$ there remain at least two substrings a^n or b^n and we also obtain a word that is not in L_n. Therefore $L_n \notin \mathcal{L}_f$*(CD-*$(n-1)$*LIN)*. □

Theorem 4.3. *Let* $f \in \{*, t, = k, \geq k, \leq k \mid k \geq 1\}$ *then* $\mathcal{L}_f$*(*CD_l*-1LIN)=LIN.*

Proof. Let L be a language in LIN then there is a linear CD grammar system Γ with one component that consists of the productions of the linear context-free grammar G with $L(G) = L$ and chain productions $A \to A$ for each nonterminal A in G. Γ generates L using an arbitrary mode in f.
Now, let Γ be a linear CD grammar system and $L = \mathcal{L}_f(\Gamma)$. If $f \in \{*, = 1, \geq 1, \leq k \mid k \geq 1\}$ we can modify the proof in [2] for unrestricted CD grammar systems and context-free grammars and obtain a linear context-free grammar that generates L.
Now, let $f = t$ and let $\Gamma = (N, T, P_1, \dots P_l, S)$ be a linear CD grammar system that generates L in t-mode. The following linear context-free grammar G generates L: $G = (\{N_i \mid 1 \leq i \leq l\} \cup \{S'\}, T, P', S')$ and $P' = \{A_i \to \alpha B_i \beta \mid A \to \alpha B \beta \in P_i, 1 \leq i \leq l\} \cup \{A_i \to A_j \mid A \notin Dom(P_i), A \in Dom(P_j) 1 \leq i, j \leq l\} \cup \{S' \to S_i \mid S \in P_i, 1 \leq i \leq l\}$.
Next, let $f = (= k)$ or $f = (\geq k)$, $k \geq 2$ and et $\Gamma = (N, T, P_1, \dots P_l, S)$ be a linear CD grammar system that generates L in $= k$-mode. The following linear context-free grammar G generates L: $G = \{N_{i,j} \mid 1 \leq i \leq k, 1 \leq j \leq l\}, T, P', S')$ and $P' = \{A_{i,j} \to \alpha B_{i+1,j} \beta \mid A \to \alpha B \beta \in P_j, 1 \leq j \leq l, 1 \leq i \leq k-1\} \cup \{A_{k,j_1} \to A_{1,j_2} \mid A \in N, 1 \leq j_1, j_2 \leq l\} \cup \{S' \to S_{1,j} \mid 1 \leq j \leq l\}$. In a similar way we can construct a linear context free grammar if $f = (\geq k)$: $G = \{N_{i,j} \mid 1 \leq i \leq k, 1 \leq j \leq l\}, T, P', S')$ and $P' = \{A_{i,j} \to \alpha B_{i+1,j} \beta \mid A \to \alpha B \beta \in P_j, 1 \leq j \leq l, 1 \leq i \leq k-1\} \cup \{A_{i,j} \to \alpha B_{i,j} \beta \mid A \to \alpha B \beta \in P_j, 1 \leq j \leq l, 1 \leq i \leq k\} \cup \{A_{k,j_1} \to A_{1,j_2} \mid A \in N, 1 \leq j_1, j_2 \leq l\} \cup \{S' \to S_{1,j} \mid 1 \leq j \leq l\}$. □

Theorem 4.4. *Let* $f \in \{= k, \geq k \mid k \geq 2\}$ *then* $\mathcal{L}_f$*(CD-mLIN)*$\subseteq \mathcal{L}(MAT_{mLIN})$

Proof. We consider the proof in the general case from [2] theorem 3.13 and 3.14. When the construction is applied to a metalinear CD grammar system the structure

of the productions is not changed and the resulting matrix grammar is metalinear and of the same width. □

With a proof related to that for the 2-normal form [3] for matrix grammars it will be possible to show that for all CD grammar systems of finite index or metalinear CD grammar systems there exists an equivalent CD grammar system working in $(= 2)$-mode that is of finite index or metalinear, respectively [1]. Hence it is likely that there exists a pumping lemma for metalinear CD grammar systems working in one of the k-modes that is easier than the one shown in this paper.

Theorem 4.5. *For the t-mode the following holds:*

1. $\mathcal{L}_t(CD_3\text{-}mLIN) = \mathcal{L}_t(CD\text{-}mLIN)$
2. $\mathcal{L}_t(CD_1\text{-}mLIN) = \mathcal{L}_t(CD_2\text{-}mLIN)$

Proof. ¿From [4] Theorem 3.1 we have the proof for 1. and in [2] Theorem 3.10 is the proof for 2. in the general case. When we apply the constructions to a metalinear CD grammar system the structure of the productions is not changed. Thus the resulting CD grammar system in 1. has only three components and is still metalinear and of the same width. The resulting context-free grammar in 2. is m-linear. □

Since metalinear CD grammar systems working in the t-mode are equivalent to metalinear $ET0L$ systems the normal form and the pumping lemma from [6] also holds for metalinear CD grammar systems working in t mode.

Theorem 4.6. $\mathcal{L}_t(CD\text{-}mLIN) = \mathcal{L}(ET0L_{mLIN})$

Proof. In [2] theorem 3.10 it is shown that $\mathcal{L}(ET0L) \subseteq \mathcal{L}_t(CDCF)$. When the construction is applied to a metalinear CD grammar system the structure of the productions is not changed and we obtain $\mathcal{L}(ET0L_{mLIN}) \subseteq \mathcal{L}_t(CD\text{-}mLIN)$.
Now we consider an m-linear CD grammar system $G = (N, T, P_1, \ldots, P_n, S)$. We construct an m-linear $ET0L$ system G' that generates $\mathcal{L}_t(G)$. $G' = (\{A^i \mid 1 \leq i \leq n \mid A \in N\} \cup \{X\} \cup T, T, \{Q_1, \ldots, Q_n\} \cup \{Q_{j,k} \mid 1 \leq k \neq j \leq n\}, S)$. The tables are defined as follows

$$\begin{aligned}
Q_i &= \{A^i \to \alpha B^i \beta \mid A \to \alpha B \beta \in P_i\} \cup \\
&\quad \{S \to A_1^i \ldots A_{m_0}^i \mid S \to A_1 \ldots A_{m_0} \in P_i\}, \\
&\quad 1 \leq i \leq n \\
Q_{j,k} &= \{A^j \to A^k \mid A \notin Dom(P_j)\} \cup \\
&\quad \{A^j \to X \mid A \in Dom(P_j)\}, \\
&\quad 1 \leq k \neq j \leq n
\end{aligned}$$

Note that G' is metalinear and of the same width as G. A derivation in G with component P_i is simulated in G' by applying Q_i one ore more times. Note that a change of superscript i of the nonterminals in G' is only possible if there is no nonterminal from $Dom(P_i)$ in the derivation string. In this way the stop conditions of the t-mode derivations in G are correctly simulated in G'. In summary we have $\mathcal{L}_t(CD\text{-}mLIN) \subseteq \mathcal{L}(ET0L_{mLIN})$ □

5 Closure Properties

Since metalinear CD grammar systems that are working in t-mode are equivalent to metalinear $ET0L$ systems, the closure properties for this mode can be found in [6]. The following proofs hold for all the derivation modes considered in this paper.

Theorem 5.1. *Let $f \in \{t, = k, \geq k \mid k \geq 2\}$, then $\mathcal{L}_f$(CD-mLIN) is closed with respect to*

1. *union*
2. *intersection with regular languages*
3. *homomorphism*
4. *inverse homomorphism.*

Proof. 1. Let L_1 and L_2 be in $\mathcal{L}_f$(CD-mLIN) and $G_1 = (N_1, T_1, P_{1,1}, \ldots, P_{1,n_1}, S_1)$ and $G_2 = (N_2, T_2, P_{2,1}, \ldots, P_{2,n_2}, S_2)$ two m-linear CD grammar systems where $\mathcal{L}_f(G_1) = L_1$ and $\mathcal{L}_f(G_2) = L_2$. W.l.o.g. let N_1 and N_2 be disjoint.

Then let S be a new symbol where $S \notin (N_1 \cup N_2)$. Construct $P'_{i,j}$ from $P_{i,j}$ in the following way: For each production $S_i \to \alpha$ in $P_{i,j}$ add a new production $S \to \alpha$ to $P'_{i,j}$. The m-linear CD grammar system $G = (N_1) \cup (N_2) \cup \{S\}, T_1 \cup T_2, P'_{1,1}, \ldots, P'_{1,n_1}, P'_{2,1}, \ldots, P'_{2,n_2}, S)$ generates $L_1 \cup L_2$.
2. Let L be a language in $\mathcal{L}_f$(CD-mLIN) and $G = (N, T, P_1, \ldots, P_n, S)$ be a grammar system with $L(G) = L$. Furthermore, let R be a regular set which is accepted by the deterministic finite automaton $M = (\Sigma, Q, \delta, q_0, F)$. Let S' be a new symbol with $S' \notin N$ and define $N' = \{[q, A, p] \mid q, p \in Q, A \in N\} \cup \{S'\}$. For all i with $1 \leq i \leq n$ and each production $p : A \to A_1 \ldots A_l, A \in N, A_1, \ldots, A_n \in N \cup T$ in P_i we define the productions $p' = \{[q, A, p] \to [q, A_1, q_1][q_1, A_2, q_2] \ldots [q_{l-1}, A_l, p] \mid q, q_1, \ldots, q_{l-1}, p \in Q\}$. We construct p'' from p' by replacing all symbols $[q, a, p]$, $q, p \in Q, a \in T$ and $\delta(q, a) = p$ with a. Furthermore, the productions containing symbols $[q, a, p]$ with $q, p \in Q, a \in T$ and $\delta(q, a) \neq p$ are deleted from p''. Let $P'_i = \bigcup_{p \in P} p''$ for $1 \leq i \leq n$. If P_i contains a production with S on its left hand side, then replace in P'_i the nonterminals $[q_0, S, f]$, $f \in F$ with S'. The m-linear CD grammar system $G' = (N', T, P'_1, \ldots, P'_n, S')$ generates $L \cap R$.
3. Let L be a language in $\mathcal{L}_f$(CD-mLIN) and $G = (N, T, P_1, \ldots, P_n, S)$ be a grammar system with $L(G) = L$. Furthermore, let $h : T \to T'^*$ be a homomorphism that is continued on T^* as usual. We construct a production set P'_i for $1 \leq i \leq n$ by replacing each production $p : A \to \alpha B \beta$ with $A, B \in N$ and $\alpha, \beta \in T^*$ in P_i by $p' : A \to h(\alpha) B h(\beta)$. The m-linear CD grammar system $G' = (N, T', P'_1, \ldots, P'_n, S)$ generates $h(L)$.
4. Let L be a language in $\mathcal{L}_f$(CD-mLIN) and $G = (N, T, P_1, \ldots, P_n, S)$ be a grammar system with $L(G) = L$. Furthermore, let $h : T' \to T^*$ be a homomorphism. W.l.o.g. $T \cap T' = \emptyset$. The language

$$\begin{aligned} K \;=\; & \{y \in (T \cup T')^* \mid y = z_0 x_1 z_1 \ldots z_{l-1} x_l z_l,\ x_1 \ldots x_l \in L(G), \\ & z_0, \ldots, z_l \in T'^2 \cup T' \cup \{\varepsilon\}\} \end{aligned}$$

is generated by the m-linear CD grammar system G' which is constructed from G as follows. For each $a \in T$ we define the set $L_a = \{xay \mid x, y \in T' \cup \{\varepsilon\}\}$. The sets L_a are finite. Now, if a production $p : A \to a_1 \dots a_{r_1} B b_1 \dots b_{r_2}$ $r_1 \geq 0, r_2 \geq 0$ $B \in N \cup \{\varepsilon\}$, $A \in N$, $a_1, \dots, a_{r_1}, b_1, \dots, b_{r_2} \in T$ occurs in a component, it is replaced by the set of productions $L_p = \{A \to a'_1 \dots a'_{r_1} B b'_1 \dots b'_{r_2} \mid a'_i \in L_{a_i}, b'_j \in L_{b_j},\ 1 \leq i \leq r_1,\ 1 \leq j \leq r_2\}$.

We consider the set

$$M = \{h(y_1)y_1h(y_2)y_2 \dots h(y_n)y_n \mid n \geq 1, y_i \in T'\}.$$

It can be seen that M is regular. The intersection between K and M can be described as follows:

$$K \cap M = \{h(y_1)y_1h(y_2)y_2 \dots h(y_n)y_n \mid h(y_1) \dots h(y_n) \in L(G)\}.$$

Now we define the homomorphism $j : (T \cup T') \to T'$, $j(a) = a$ if $a \in T'$ and $j(a) = \varepsilon$ if $a \in T$. It is easy to see that $j(K \cap M) = h^{-1}(L(G))$. Since $\mathcal{L}_f$*(CD-mLIN)* is closed under intersection with regular sets and homomorphism, it follows that $\mathcal{L}_f$*(CD-mLIN)* is closed under inverse homomorphism. □

Theorem 5.2. *Let $f \in \{t, = k, \geq k \mid k \geq 2\}$, then $\mathcal{L}_f$(CD-mLIN) is not closed with respect to*

1. *concatenation,*
2. *Kleene-closure,*
3. *intersection,*
4. *complement.*

Proof. 1. With [7] Theorem 4.1 we can show that $L_m = \{a^n b^n \mid n \geq 0\}^{2m-1}$ is in $\mathcal{L}_f$*(CD-mLIN)* for $f \in \{t, = k, \geq k \mid k \geq 2\}$. But $L_m L_m$ contains words $w = (a^h b^h)^{4m-2}$. With the technique from Lemma 4.1 it can be shown that $L_m L_m$ is not in $\mathcal{L}_f$*(CD-mLIN)*.
2. follows from 1.
3. Consider the languages $L_1 = \{a^* b^* (a^n b^n)^m \mid n \geq 1\}$ and $L_2 = \{(a^n b^n)^m a^* b^* \mid n \geq 1\}$. L_1 can be generated by the grammar system

$$\begin{aligned}
G_1 &= \{\{S, A, B, C\}, \{a, b\}, P_1, P_2, P_3, P_4, P_5, S\} \\
P_1 &= \{S \to CA^k\} \\
P_2 &= \{C \to aC, C \to aB, B \to bB, B \to bA\} \\
P_3 &= \{A \to aA'b\} \\
P_4 &= \{A' \to A\} \\
P_5 &= \{A \to ab\}
\end{aligned}$$

using the t-mode. With similar methods as in [7] Theorem 4.1 an m-linear CD grammar system that accepts L_1 in the $\{= k, \geq k \mid k \geq 2\}$-modes can be constructed. Now, we change $S \to CA^k$ to $S \to A^k C$ and obtain an m-linear CD grammar system

G_2 that generates L_2. With similar methods as in [7] Theorem 4.1 an m-linear CD grammar system that accepts L_2 in the $\{=k, \geq k \mid k \geq 2\}$-modes can be constructed. The intersection between the two languages is $L_1 \cap L_2 = \{(a^n b^n)^{m+1}\}$. We know from Theorem 4.2 that this language is not in $\mathcal{L}_f$*(CD-mLIN)*.
4. With De Morgan's Theorem and the fact that $\mathcal{L}_f$*(CD-mLIN)* is closed under union but not closed under intersection we obtain that $\mathcal{L}_f$*(CD-mLIN)* is not closed with respect to complement. □

6 Conclusion

We have defined classes of metalinear CD grammar systems and intensely investigated their generative capacity. To obtain the results a pumping lemma was shown. The following holds

$$\begin{aligned} LIN &= \mathcal{L}_f(CD\text{-}LIN) \subset \mathcal{L}_f(CD\text{-}2LIN) \subset \ldots \\ &\subset \mathcal{L}_f(CD\text{-}mLIN) \subset \mathcal{L}_f(CD\text{-}(m+1)LIN) \subset \ldots . \end{aligned}$$

Furthermore it is

$$\begin{aligned} \mathcal{L}_t(CD_1\text{-}mLIN) = \mathcal{L}_t(CD_2\text{-}mLIN) &\subset \\ \mathcal{L}_t(CD_3\text{-}mLIN) &= \mathcal{L}_t(CD\text{-}mLIN) = \mathcal{L}(ET0L_{mLIN}). \end{aligned}$$

and

$$\mathcal{L}_f(CD\text{-}mLIN) \subseteq \mathcal{L}(MAT_{mLIN}), \ f \in \{=k, \geq k \mid k \geq 2\}.$$

We have shown that there is a context-free language that can not be generated with metalinear CD grammar systems although many context-sensitive languages can be. At last we have investigated the closure properties of metalinear CD grammar systems. The researched language classes are closed under union, intersection with regular languages, homomorphism and inverse homomorphism. They are not closed under concatenation, Kleene closure intersection and complement.

Metalinear CD grammar systems were defined to obtain grammars that are more powerful than context free grammars but have the simplest possible derivation structure. We got a deep insight into how the width of a metalinear CD grammar system influences its generative capacity. We have also obtained a pumping lemma which allows us to decide for certain languages if they are not generated by metalinear CD grammar systems. To complete the research of the generative power it would be interesting to know how the number of components influence the generative capacity of the metalinear CD grammar systems in $(=k)$-mode and $(\geq k)$-mode.

References

[1] H. Bordihn, B. Sunckel, On the Number of Active Symbols in CD grammar systems. Unpublished manuscript.

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

[3] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory.* Springer, Berlin, 1989.

[4] J. Dassow, G. Păun , G. Rozenberg, Grammar Systems. In: G. Rozenberg, A. Salomaa (ed.), *Handbook of Formal Languages Vol. 2.* Springer-Verlag, Berlin, 1997, 155–213.

[5] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, London, 1979.

[6] G. Rozenberg, D. Vermeir, On metalinear ET0L systems. *Fundamenta Informaticae* **3(1)** (1980), 15–36.

[7] B. Sunckel, On the descriptional complexity of metalinear CD grammar systems, In: L. Ilie, D. Wotschke (ed.), *Pre-proceedings of the Sixth International Workshop on Descriptional Complexity of Formal Systems (DCFS 2004)*, Report No. 619, Department of Computer Science, The University of Western Ontario, London, Ontario, Canada, 2002.

Author Index

Arun Prasath A., 15

Bel Enguix G., 30
Bensch S., 50
Bernardini F., 66
Bordihn H., 50, 78, 95

Choudhary A., 287
Cojocaru L., 113, 126
Csuhaj-Varjú E., 141

Dassow J., 158
Dersanambika K. S., 170

Freund R., 179

Gheorghe M., 66
Grando M. A., 200

Holzer M., 78

Jiménez López M. D., 30

Kari L., 212
Kelemen J., 225
Kelemenová A., 240
Krithivasan K., 15, 170, 287

Losseva E., 212

Mitrana V. 200
Martín-Vide C. 126

Oswald M., 179

Păun Gh., 255

Roslin Sagaya Mary A., 276

Sathana Krishnan G. N., 287
Sebestyén P., 300
Sosík P., 212, 300
Subramanian K. G., 170, 276
Sunckel B., 313

Thierrin G., 212
Tupý M., 240

Vaszil Gy., 95