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 $X \rightarrow u(tar)$, where $X \rightarrow u$ 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 $X \rightarrow u$ 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 $\Rightarrow_i$, $1 \leq i \leq n$. 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

$$\begin{aligned} L_t(\Gamma) &= \{w \in T^* \mid S \Rightarrow_{i_1}^t w_1 \Rightarrow_{i_2}^t \ldots \Rightarrow_{i_k}^t w_k = w, \\ &\quad \text{for some } 1 \leq i_1, i_2, \ldots, i_k \leq m,\ k \geq 1\}. \end{aligned}$$

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 $m \geq 1$ 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 $CD_m(t), m \geq 1$) 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

$$\Pi = (V, T, \mu, (w, i_0), R_1, \dots, R_m),$$

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 $H = \{1, 2, \dots, m\}$);
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. $R_1, \dots, R_m$ 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 $a^ncb^na^mdb^m$ 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 $a^{n+1}c'b^{n+1}a^mdb^m$ $a^{n+1}c'b^{n+1}a^{m+1}d'b^{m+1}$. 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 $a^ncb^na^md'b^m$ 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^nb^n of the string. Therefore, $L(\Pi) = \{a^nb^na^mb^m \mid n \geq m \geq 1\}$. 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^nb^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 $L(\Pi) = \{a^nba^na^mca^m \mid m > n \geq 1\}$.

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).

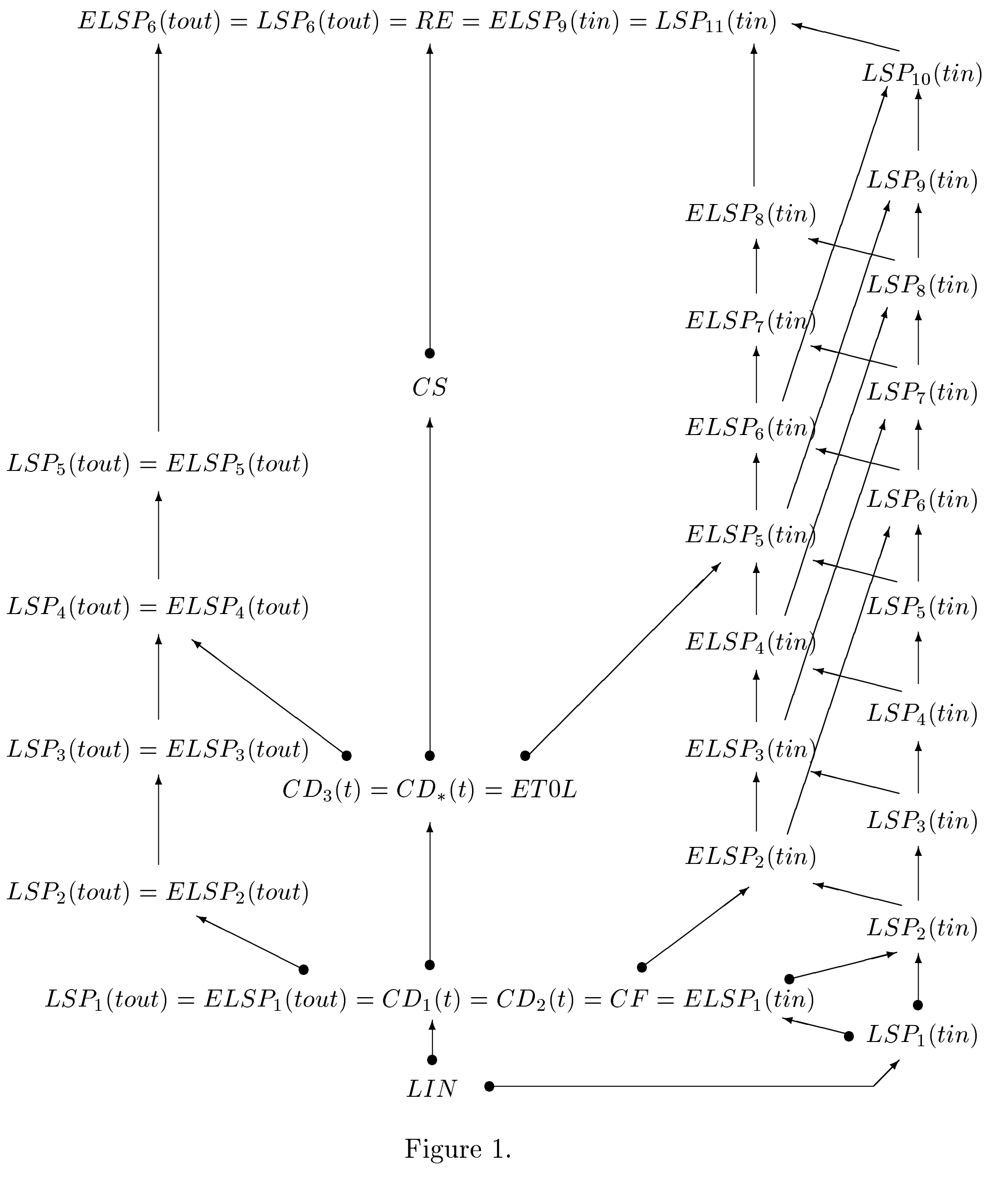
$ELSP_6(tout) = LSP_6(tout) = RE = ELSP_9(tin) = LSP_{11}(tin)$
$LSP_{10}(tin)$
$LSP_9(tin)$
$ELSP_8(tin)$
$LSP_8(tin)$
$ELSP_7(tin)$
CS
$LSP_7(tin)$
$ELSP_6(tin)$
$LSP_5(tout) = ELSP_5(tout)$
$LSP_6(tin)$
$ELSP_5(tin)$
$LSP_4(tout) = ELSP_4(tout)$
$LSP_5(tin)$
$ELSP_4(tin)$
$LSP_4(tin)$
$LSP_3(tout) = ELSP_3(tout)$
$ELSP_3(tin)$
$CD_3(t) = CD_*(t) = ET0L$
$LSP_3(tin)$
$ELSP_2(tin)$
$LSP_2(tout) = ELSP_2(tout)$
$LSP_2(tin)$
$LSP_1(tout) = ELSP_1(tout) = CD_1(t) = CD_2(t) = CF = ELSP_1(tin)$
$LSP_1(tin)$
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

$$\begin{aligned}
\Pi &= (V, V, \mu, (X_0A_0, 1), R_1, \dots, R_6), \text{ where}\\
V &= \{X, X', X'', X''' \mid X \in N_1 \cup \{f\}\} \cup T\\
&\cup \{X_{i,j}, X'_{i,j}, X''_{i,j}, X'''_{i,j}, \bar{X}_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\}\\
&\cup \{A_{i,j}, A'_{i,j}, A''_{i,j}, A'''_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\}\\
&\cup \{X^{(j)}_i \mid m_i : (X \to Y, B^{(j)} \to \#), 1 \le i \le k_j, j = 1, 2\},\\
\mu &= [_1[_2[_3[_4\ ]_4]_3]_2[_5\ ]_5[_6\ ]_6]_1,
\end{aligned}$$

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 $A \to A_{i,0}(in)$, $X \to X^{(s)}_i(in)$. 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	$A \to A_{i,0}(in)$	$X \to X_i^{(s)}(in)$
	$X''_{i,j} \to X'''_{i,j}(in)$	
	$X_{i,j} \to X_{i,j}(in)$	
	$Y''' \to Y$	
R_2	$X \to \bar{X}_{i,0}(in)$	$X_i^{(s)} \to X_i^{(s)}$
	$X'''_{i,j} \to X_{i,j}(in)$	
	$A''_{i,j} \to A'''_{i,j}$	
	$A_{i,0} \to A_{i,0}$	
	$Y'' \to Y'''$	
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	$X_{i,0} \to X_{i,0}$	$X_i^{(1)} \to Y$
	$A_{i,0} \to A_{i,0}$	$X_i^{(2)} \to X_i^{(2)}$
	$X'''_{i,j} \to X'''_{i,j}$	$B^{(1)} \to B^{(1)}$
R_6	$X_{i,0} \to X_{i,0}$	$X_i^{(2)} \to Y$
	$A_{i,0} \to A_{i,0}$	$X_i^{(1)} \to X_i^{(1)}$
	$X'''_{i,j} \to X'''_{i,j}$	$B^{(2)} \to B^{(2)}$

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 $Y'w_1A'_{i,i'}w_2$; we pass to $Y''w_1A''_{i,i'}w_2$, which is sent to region 2, where we produce $Y'''w_1A'''_{i,i'}w_2$ 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 $\bar{Y}_{s,0}w_1A'''_{i,i'}w_2$ 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

$$\begin{aligned}
\Pi &= (V, T, \mu, (X_0A_0, 1), R_1, \ldots, R_9), \text{ where} \\
V &= \{X, X', X'', X''' \mid X \in N_1 \cup \{f\}\} \cup T \\
&\cup \{X_{i,j}, X'_{i,j}, X''_{i,j}, X'''_{i,j}, \bar{X}_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
&\cup \{A_{i,j}, A'_{i,j}, A''_{i,j}, A'''_{i,j} \mid m_i : (X \to Y, A \to u), 1 \le i \le k, 0 \le j \le k\} \\
&\cup \{X_i^{(j)} \mid m_i : (X \to Y, B^{(j)} \to \#), 1 \le i \le k_j, j = 1, 2\}, \\
\mu &= [_1[_2[_3\ ]_3[_6[_7\ ]_7]_6[_8[_9\ ]_9]_8]_2[_4[_5\ ]_5]_4]_1,
\end{aligned}$$

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. $\square$

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 $c_{j_1}c_{j_2}\dots c_{j_s}$ indicating the clauses in γ which contain x_i, and by $true(\gamma, f_i)$ the string $c_{j_1}c_{j_2}\dots c_{j_s}$ 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 $0, 1.1, 1.2, 1.3, \dots, i.1, i.2, i.3, \dots, n.1, n.2, n.3, 1, 1', 1'', \dots, m, m', m''$, $m+1$)

$$
\begin{array}{rcl}
\Pi &=& (N, K, T, M_0, M_{1.1}, \dots, M_{m''}, M_{m+1}, R_0, R_{1.1}, \dots, R_{m''}, R_{m+1}, m+1),\\
N &=& \{a_i \mid 1 \leq i \leq 3n + 2m + 1\}\\
&\cup& \{a'_i,\ a''_i \mid 1 \leq i \leq 2n\}\\
&\cup& \{t_i,\ f_i \mid 1 \leq i \leq n\}\\
&\cup& \{c_i \mid 1 \leq i \leq m\}\\
&\cup& \{b\},\\
K &=& \{Q_0,\ Q_{m+1}\}\\
&\cup& \{Q_{i.1},\ Q_{i.2},\ Q_{i.3} \mid 1 \leq i \leq n\}\\
&\cup& \{Q_i,\ Q_{i'},\ Q_{i''} \mid 1 \leq i \leq m\},\\
T &=& \{d\},\\
M_0 &=& \{b\},\\
M_{i.1} &=& M_{i.2} = M_j = M_{j'} = M_{m+1} = \{a_1\},\ \text{for all } 1 \leq i \leq n, 1 \leq j \leq m,\\
M_{i.3} &=& \{a'_i,\ a''_i\},\ \text{for all } 1 \leq i \leq n,\\
M_{i''} &=& \{d\},\ \text{for all } 1 \leq i \leq m,\\
R_0 &=& \emptyset,\\
R_{1.1} &=& \{a_1 \to Q_0,\ b \to t_1 b\},\\
R_{1.2} &=& \{a_1 \to Q_0,\ b \to f_1 b\},\\
R_{1.3} &=& \{a'_1 \to a'_2,\ a''_1 \to a''_2,\ a'_2 \to Q_{1.1},\ a''_2 \to Q_{1.2}\},\\
R_{i.1} &=& \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(i-1)\}\\
&\cup& \{a_{2i-1} \to Q_{i-1.3},\ b \to t_i b\},\ \text{for all } i = 2, 3, \dots, n-1,\\
R_{i.2} &=& \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(i-1)\}\\
&\cup& \{a_{2i-1} \to Q_{i-1.3},\ b \to f_i b\},\ \text{for all } i = 2, 3, \dots, n-1,\\
R_{i.3} &=& \{a'_j \to a'_{j+1},\ a''_j \to a''_{j+1} \mid 1 \leq j \leq 2i-1\}\\
&\cup& \{a'_{2i} \to Q_{i.1},\ a''_{2i} \to Q_{i.2}\},\ \text{for all } i = 2, 3, \dots, n-1,\\
R_{n.1} &=& \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(n-1)\}\\
&\cup& \{a_{2n-1} \to Q_{n-1.3},\ b \to t_n\},\\
R_{n.2} &=& \{a_j \to a_{j+1} \mid 1 \leq j \leq 2(n-1)\}\\
&\cup& \{a_{2n-1} \to Q_{n-1.3},\ b \to f_n\},\\
R_{n.3} &=& \{a'_j \to a'_{j+1},\ a''_j \to a''_{j+1} \mid 1 \leq j \leq 2n-1\}\\
&\cup& \{a'_{2n} \to Q_{n.1},\ a''_{2n} \to Q_{n.2}\}\\
&\cup& \{t_i \to true(\gamma, t_i),\ f_i \to true(\gamma, f_i) \mid 1 \leq i \leq n\},\\
R_1 &=& \{a_j \to a_{j+1} \mid 1 \leq j \leq 3n\}
\end{array}
$$

$$
\begin{aligned}
& \cup && \{a_{3n+1} \to Q_{n.3},\ c_1 \to Q_{1''}\},\\
R_{1'} & = && \{a_j \to a_{j+1} \mid 1 \le j \le 3n+1\}\\
& \cup && \{a_{3n+2} \to Q_1\},\\
R_i & = && \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2(i-1)\}\\
& \cup && \{a_{3n+2(i-1)+1} \to Q_{i-1},\ c_i \to Q_{i''}\},\ \text{for all } i = 2,3,\ldots,m,\\
R_{i'} & = && \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2(i-1)+1\}\\
& \cup && \{a_{3n+2(i-1)+2} \to Q_i\},\ \text{for all } i = 2,3,\ldots,m,\\
R_{i''} & = && \emptyset,\ \text{for all } i = 1,2,\ldots,m,\\
R_{m+1} & = && \{a_j \to a_{j+1} \mid 1 \le j \le 3n+2m\}\\
& \cup && \{a_{3n+2m+1} \to Q_m\}.
\end{aligned}
$$

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.