Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Theoretical Computer Science 301 (2003) 201 – 215

Theoretical
Computer Science

www.elsevier.com/locate/tcs

On three variants of rewriting P systems

Claudio Ferretti[a], Giancarlo Mauri[a], Gheorghe Păun[b], Claudio Zandron[a,*]

[a]Dipartimento di Informatica, Sistemistica e Comunicazione, Universitá di Milano-Bicocca, Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
[b]Institute of Mathematics of the Romanian Academy, P.O. Box 1-764, 70700 Bucureşti, Romania

Received 19 March 2002; received in revised form 1 June 2002; accepted 19 June 2002
Communicated by G. Rozenberg

Abstract

We continue here the study of P systems with string objects processed by rewriting rules, by investigating some questions which are classic in formal language theory: leftmost derivation, conditional use of rules (permitting and forbidding conditions), relationships with language families in Chomsky and Lindenmayer hierarchies.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Membrane computing; Regulated rewriting; Chomsky hierarchy; Lindenmayer systems

1. Introduction

One of the main classes of P systems is that where the objects are described by strings and the evolution rules are based on string processing operations, in particular, on rewriting. The reader is referred to, e.g., [1–3,5,9–13] (an up-to-date bibliography of the area can be found at the internet web address http://bioinformatics.bio.disco.unimib.it/psystems).

In the framework of P systems, in order to get characterizations of recursively enumerable languages, various additional features are considered, such as a priority relation on the set of rules from each membrane (see [14]) or the possibility to control the membrane permeability (see [17]). We consider here three further variants of such

* Corresponding author. Tel.: +39-02-64487875; fax: +39-02-64487839.
E-mail addresses: ferretti@disco.unimib.it (C. Ferretti), mauri@disco.unimib.it (G. Mauri), gpaun@imar.ro (G. Păun), zandron@disco.unimib.it (C. Zandron).

PII: S0304-3975(02)00581-9

systems, which are classic in formal language theory. In the case of imposing that each string is rewritten in the *leftmost* possible position, we get a characterization of recursively enumerable languages by systems with four membranes. Two membranes are shown to suffice in the case of using *forbidding* conditions associated with rules, in the form of symbols which should not be present in the string to be rewritten. Somewhat surprisingly, the case of *permitting* conditions (certain symbols should be present in the rewritten string) only leads to a characterization of languages generated by matrix grammars without appearance checking, hence this restriction does not increase substantially the power of rewriting P systems. In fact, rewriting P systems without restrictions in the use of the rules and with at least four membranes already characterize the family of matrix languages; two membranes are enough for generating such a family if permitting conditions are allowed.

2. Language theory prerequisites

In this section we introduce some formal language theory notions and notations which will be used in this paper; for further details we refer to [16].

For an alphabet V, by V^* we denote the set of all strings over V, including the empty one, denoted by λ. By CF, RE we denote the families of context-free and of recursively enumerable languages, respectively, while $E0L$ and $ET0L$ denote the families of languages generated by extended interactionless Lindenmayer ($E0L$) systems and by extended tabled $0L$ systems ($ET0L$ systems). It is known that $CF \subset E0L \subset ET0L \subset RE$, all inclusions being proper.

For each recursively enumerable language L there exists a type-0 grammar in the *Geffert normal form* [7], $G = (\{S, A, B, C\}, T, P, S)$, such that $L(G) = L$ and $P = P_{\rm cf} \cup P_\lambda$, where $P_{\rm cf}$ only contains context-free productions of the form $S \to w$ with $w \in (\{S, A, B, C\} \cup T)^*$ and P_λ only contains the unique (non-context-free) production $ABC \to \lambda$.

In the proofs from the next sections we need the notion of a *matrix grammar with appearance checking*; such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geqslant 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Rightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*$, $1 \leqslant i \leqslant n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leqslant i \leqslant n$, either $w_i = w_i' A_i w_i''$, $w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = {}_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied—one says that these rules are applied in the *appearance checking* mode.)

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. The family of languages of this form is denoted by $MAT_{\rm ac}$. When $F = \emptyset$ (hence we do not use the appearance checking feature), the generated family is denoted by MAT.

It is known that $CF \subset MAT \subset MAT_{\rm ac} = RE$, all inclusions being proper. All one-letter languages in the family MAT are regular, see [8]. Moreover, $E0L$ is incomparable with

MAT (hence $ET0L - MAT \neq \emptyset$, but it is not known whether also $MAT - ET0L$ is non-empty).

A matrix grammar $G=(N,T,S,M,F)$ is said to be in the *binary normal form* if $N=N_1 \cup N_2 \cup \{S,\#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

(1) $(S \to ZB)$, with $Z \in N_1, B \in N_2$,
(2) $(X \to Y, A \to x)$, with $X,Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leqslant 2$,
(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 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; # is called 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.

According to Lemma 1.3.7 in [4], for each matrix grammar there is an equivalent matrix grammar in the binary normal form. For an arbitrary matrix grammar $G=(N,T,S,M,F)$, let us denote by $ac(G)$ the cardinality of the set $\{A \in N \mid A \to \alpha \in F\}$. From the construction in the proof of Lemma 1.3.7 in [4] one can see that if we start from a matrix grammar G and we get the grammar G' in the binary normal form, then $ac(G')=ac(G)$.

In [6] it is proved that each recursively enumerable language can be generated by a matrix grammar G such that $ac(G) \leqslant 2$. Consequently, to the properties of a grammar G in the binary normal form we can add the fact that $ac(G) \leqslant 2$. We will say that this is *the strong binary normal form* for matrix grammars.

3. Rewriting P systems

We introduce here the class of P systems with string objects processed by rewriting and without any other additional feature, then we will add further ingredients, able to increase the generative power. (As usual, a membrane structure is represented by a string of labeled parentheses, and with each membrane we associate a *region*, which is referred to by the label of the membrane.)

An *extended rewriting P system* (ERP system) of degree $m \geqslant 1$, is a construct

$$\Pi = (V, T, \mu, M_1, \ldots, M_m, R_1, \ldots, R_m),$$

where:

(1) V is the alphabet of the system;
(2) $T \subseteq V$ is the *terminal alphabet*;
(3) μ is a membrane structure with m membranes (in this section, we assume that the membranes are injectively labeled by $1, 2, \ldots, m$);
(4) $M_1, \ldots, M_m$ are finite languages over V, representing the strings initially present in the regions $1, 2, \ldots, m$ of the system;

(5) $R_1, \ldots, R_m$ are finite sets of *rules* of the form $X \to (u, tar)$, with $X \in V, u \in V^*, tar \in \{here, out, in\}$, associated with the regions of μ. Often, the indication *here* is omitted.

A system is said to be *non-extended* if $V = T$.

From a configuration of the system we pass to another configuration by rewriting, in each region of the system, each string which can be rewritten by a rule from that region, and following the target indication of the used rule. For instance, when rewriting a string x_1Xx_2 by a rule $X \to (u, tar)$ we get the string x_1ux_2, which will be communicated to the membrane indicated by *tar* (*here* means that the string remains in the same region, *out* means that the string will exit, and *in* means that the string is sent to one of the directly lower membranes, if any exists, otherwise the rule cannot be applied). Note that all strings are processed at the same time, but each string is rewritten by only one rule—or by none, if no rule can be applied to it.

A sequence of transitions forms a computation and the result of a halting computation is the set of strings over T sent out of the system during the computation. In the case of non-extended systems, all strings sent out are accepted. A computation which never halts yields no result. A string which remains inside the system or, in the extended case, which exits but contains symbols not in T does not contribute to the generated language. The language generated in this way by a system Π is denoted by $L(\Pi)$. The family of all languages of this form, generated by P systems of degree at most $m, m \geqslant 1$, is denoted by $ERP_m(free)$.

The reader can easily see that every language in $ERP_m(free)$ can also be generated by a matrix grammar without appearance checking (see a stronger result in Theorem 10), hence it is necessary to increase the power of such P systems by adding further features. The next three sections are devoted to such a topic.

4. Leftmost rewriting in P systems

We consider here a restriction in the use of rules of a rewriting P system: any string is rewritten in the leftmost position which can be rewritten by a rule from its region. That is, we examine the symbols of the string, one by one, from left to right; the first symbol which can be rewritten by a rule from the region of the string is rewritten. If there are several rules with the same left-hand symbol, then any of them is chosen.

We denote by $L_{left}(\Pi)$ the language generated by a system Π in this way and by $ERP_m(left)$, $m \geqslant 1$, we denote the family of all such languages, generated by systems with at most m membranes. If the degree of the systems is not bounded, then we replace the subscript m by $*$.

In the regulated rewriting area it is known that the leftmost derivation (of a different type: only the first rule of a matrix is used in the leftmost manner—see precise definitions in [4]) increases the power of matrix grammars. This makes expected the result from Theorem 3 below (the main difficulty is to find a reasonably small number of membranes which suffice).

Before proving this assertion, let us give some preliminary results. Directly from the definitions, we have ($[E]$ means that the relation is true both with E in both its members and with E in none of them):

Lemma 1. (1) $RP_m(left) \subseteq ERP_m(left)$, $m \geqslant 1$.
(2) $[E]RP_m(left) \subseteq [E]RP_{m+1}(left)$, $m \geqslant 1$.

Lemma 2. $ERP_m(left) \subseteq RP_{m+1}(left), m \geqslant 1$.

Proof. Given $\Pi = (V, T, \mu, M_1, \ldots, M_m, R_1, \ldots, R_m)$, we construct a system $\Pi' = (V \cup \{Y, Z\}, V \cup \{Y, Z\}, [_0\mu]_0, \emptyset, M'_1, \ldots, M'_m, R_0, R_1, \ldots, R_m)$, with

$$M'_i = \{wY \mid w \in M_i\},\ 1 \leqslant i \leqslant m,$$

$$R_0 = \{a \to Z \mid a \in V - T\} \cup \{Z \to Z,\ Y \to (\lambda, out)\}.$$

It is easy to see that $L_{left}(\Pi') = L_{left}(\Pi)$: a string can exit the skin membrane of Π' only by using the rule $Y \to (\lambda, out)$. Because Y appears in the right-hand end of the string, this means that the whole string is checked whether or not it contains any symbol not in T (in such a case, the trap symbol Z is introduced and the computation never stops). □

Theorem 3. $RE = RP_5(left) = ERP_4(left)$.

Proof. According to the previous lemma and to Turing–Church thesis, we only have to prove the inclusion $RE \subseteq ERP_4(left)$.

Let $G = (\{S, A, B, C\}, T, P, S)$ be a grammar in the Geffert normal form.

We construct the extended P system of degree 4

$$\Pi = (V, T, \mu, M_1, M_2, M_3, M_4, R_1, R_2, R_3, R_4)$$

with

$$\begin{aligned}
V &= \{E, \bar{E} \mid E \in \{A, B, C, S, Y\} \cup T\} \cup \{D, F, Z\},\\
\mu &= [_1[_2[_3]_3]_2[_4]_4]_1,\\
M_1 &= \{SY\},\ M_2 = M_3 = M_4 = \emptyset,\\
R_1 &= \{E \to \bar{E} \mid E \in \{A, B, C, S\} \cup T\}\\
&\cup\ \{S \to w \mid S \to w \in P_{\mathrm{cf}}\}\\
&\cup\ \{A \to (F, in), Y \to (\bar{Y}, in)\}\\
&\cup\ \{a \to (a, out) \mid a \in T\} \cup R'_1,\\
R'_1 &= \begin{cases} \{Y \to (\lambda, out)\} & \text{if } \lambda \in L(G) \text{ and}\\ \emptyset & \text{if } \lambda \notin L(G), \end{cases}
\end{aligned}$$

$$\begin{aligned} R_2 &= \{B \to (\lambda, in), D \to (\lambda, out), F \to \lambda\} \\ &\cup \{E \to Z \mid E \in \{A, C, S, Z\} \cup T\}, \\ R_3 &= \{C \to (D, out)\} \cup \{E \to Z \mid E \in \{A, B, S, Z\} \cup T\}, \\ R_4 &= \{\bar{E} \to E \mid E \in \{A, B, C, S\} \cup T\} \\ &\cup \{F \to Z, \bar{Y} \to (Y, out), \bar{Y} \to (\lambda, out), Z \to Z\}. \end{aligned}$$

Initially, we start with the string SY in membrane 1; assume that here in general we have a string $\bar{v}wU$ with $v, w \in (\{A,B,C,S\} \cup T)^*$ and $U \in \{\lambda, Y\}$.

For $v = U = \lambda$ we can send out a terminal word $w \in T^+$, $w = aw'$ for some $a \in T$, by applying the rule $a \to (a, out)$. If $v \neq \lambda$, an application of a rule $a \to (a, out)$, $a \in T$, does not yield a terminal word outside the system, hence this string is "lost". If also $\lambda \in L(G)$, then $Y \to (\lambda, out)$ yields the desired result λ outside the system for $v = w = \lambda$ and $U = Y$. The rule $Y \to (\lambda, out)$ in the leftmost derivation mode can only be applied in membrane 1 to a string of the form $\bar{v}Y$; for $v \neq \lambda$ the application of $Y \to (\lambda, out)$ only yields the non-terminal (and therefore "useless") string $\bar{v}$ outside the system.

The rule $Y \to (\bar{Y}, in)$ in the leftmost derivation mode can only be applied in membrane 1 to a string of the form $\bar{v}Y$, too. If the resulting string $\bar{v}\bar{Y}$ lands in membrane 2, then the computation stops without having generated a result, whereas in membrane 4 from $\bar{v}\bar{Y}$ we obtain $v\bar{Y}$ by applying the rules $\bar{E} \to E$ for $E \in \{A,B,C,S\} \cup T$. The rule $\bar{Y} \to (Y, out)$ sends out the string vY, whereas the application of the rule $\bar{Y} \to (\lambda, out)$ yields only v in membrane 1.

The main idea of the simulation of the productions of G by leftmost applications of rules in Π is the following:

From left to right, the symbols from $\{A,B,C,S\} \cup T$ are barred. At any time, whenever possible, we may apply a rule $S \to w$, for $S \to w \in P_{\mathrm{cf}}$, in order to simulate a context-free production from G.

To start the simulation of the unique non-context-free production $ABC \to \lambda$, we choose the rule $A \to (F, in)$. If a string of the form $\bar{v}Fw'U, v, w' \in (\{A,B,C,S\} \cup T)^*$ and $U \in \{\lambda, Y\}$, lands in membrane 4, we obtain $vFw'U$ by applying the rules $\bar{E} \to E, E \in \{A,B,C,S\} \cup T$, and finally $vZw'U$, a string containing the trap symbol Z, which leads to a non-halting computation in an infinite loop with iterated applications of the rule $Z \to Z$. If, as is necessary for a correct simulation of the production $ABC \to \lambda, \bar{v}Fw'U$ appears in membrane 2, the next symbol in w' must be a B, otherwise, after the application of the rule $F \to \lambda$, either one of the rules $E \to Z, E \in \{A,C,S\} \cup T$, leads to a non-halting computation because of the rule $Z \to Z$, or, if we have reached the end (possibly the end symbol Y) of the current string, the computation stops without having yielded a terminal result.

If $w' = Bw''$, then we get $\bar{v}w''U$ in membrane 3. Here, if w'' does not begin with a C, either the rules $E \to Z, E \in \{A,B,S\} \cup T$, again lead to a non-halting computation or else the computation stops without having produced a terminal string. If $w'' = Cw'''$, then we obtain $\bar{v}Dw'''U$ in membrane 2, where the application of the rule $D \to (\lambda, out)$

leads to the string $\bar{v}w'''U$ in membrane 1; hence, in total, we have obtained $\bar{v}w'''U$ from $\bar{v}ABCw'''U$.

If, by the applications of the rules $E \to \bar{E}$, $E \in \{A,B,C,S\} \cup T$, the end of the underlying string is reached, i.e., we have obtained $\bar{v}U, v \in (\{A,B,C,S\} \cup T)^*$ and $U \in \{\lambda, Y\}$, then the computation stops without yielding a terminal result, if $U = \lambda$. By applying $Y \to (\bar{Y}, in)$ to $\bar{v}Y$, we obtain $\bar{v}\bar{Y}$ in membrane 2 or in membrane 4. In membrane 2, the computation halts, because no rule can be applied. In membrane 4, we get $v\bar{Y}$ by applications of the rules $\bar{E} \to E, E \in \{A,B,C,S\} \cup T$. Finally, by applying $\bar{Y} \to (Y, out)$ or $\bar{Y} \to (\lambda, out)$, we obtain vY or v, respectively, in membrane 1.

In sum, we can correctly simulate each production from G correctly; derivations in G yielding a terminal word $w \in T^*$ correspond with halting computations in Π which in the last step send out w. All other computations in Π either halt without having sent out a terminal string or else enter an infinite loop with the trap symbol Z. Hence, we conclude $L(\Pi) = L(G)$. □

It is an *open problem* whether or not the previous result is optimal, or the number of used membranes can be decreased.

5. Rewriting P systems with less than four membranes

As seen in the previous section, four membranes suffice to get universality. In this section we are going to give some results with respect to rewriting P systems with less than four membranes.

Theorem 4. $ET0L \subset ERP_3(left)$.

Proof. Each language $L \in ET0L$ can be generated by an $ET0L$ system with only two tables, $G = (V, T, w, P_1, P_2)$. Moreover, each derivation starts by using the first table and ends by using the second table; always, the second table is used only once, while the first table may be used several times in a row (see the proof of Theorem V.1.3 in [15]). It is also important to remember that each table in an $ET0L$ system is *complete* (a rule $a \to x$ exists in each table for each symbol $a \in V$). We construct the following P system:

$$\Pi = (V', T, [_1[_2[_3\ ]_3]_2]_1, \emptyset, wY, \emptyset, R_1, R_2, R_3),$$

$$V' = V \cup \{a' \mid a \in V\} \cup \{Y\},$$

$$R_1 = \{a' \to x \mid a \to x \in P_2\}$$
$$\cup \{Y \to (Y, in),\ Y \to (\lambda, out)\},$$

$$R_2 = \{a \to h(x) \mid a \to x \in P_1\}$$
$$\cup \{Y \to (Y, out),\ Y \to (Y, in)\},$$

$$R_3 = \{a' \to a \mid a \in V\}$$
$$\cup \{Y \to (Y, out)\},$$

where h is the morphism defined by $h(a) = a', a \in V$.

In membrane 2 we simulate the use of table P_1, applying the rules to each symbol, from left to right, with the introduced symbols being primed; when we can also rewrite the marker Y, the string is sent either to membrane 1 or to membrane 3. In membrane 3 we remove the primes and return the string to membrane 2, hence we can simulate again the first table. In membrane 1 we simulate the use of table P_2, starting from primed symbols and leading to a string composed of non-primed symbols. When all symbols are rewritten, we can either iterate the procedure, by using the rule $Y \to (Y, in)$, or we can send the string out, by using the rule $Y \to (\lambda, out)$. Consequently, $L_{left}(\Pi) = L(G)$, which proves the inclusion $ET0L \subseteq ERP_3(left)$.

This inclusion is proper. Actually, a stronger assertion is true: the family $RP_3(left)$ contains languages which are not in $ET0L$.

Indeed, let us consider the system

$$\Pi = (V, V, [_1[_2[_3\ ]_3]_2]_1, \{cfd\}, \emptyset, \emptyset, R_1, R_2, R_3),$$

$$V = \{a, a', b, b', b'', c, c', c'', d, f\},$$

$$R_1 = \{f \to fa'b',\ f \to a'b',\ a' \to (a, in),\ c'' \to (\lambda, out)\},$$

$$R_2 = \{b' \to bb'',\ d \to (d, in),\ c' \to (c, out),\ c' \to (c'', out)\},$$

$$R_3 = \{c \to c',\ b'' \to b',\ d \to (d, out)\}.$$

We start by producing a string $c(a'b')^n d$, $n \geqslant 1$, in membrane 1; after removing the symbol f, we have to replace the leftmost a' by a and the string is sent to membrane 2. Here, each b' introduces an occurrence of b and gets one more prime; when all occurrences of b' have been rewritten, we can use the rule $d \to (d, in)$. The string is sent to membrane 3, where all b'' are replaced by b', and the string is sent back to membrane 2 (with c replaced by c'). If in membrane 2 we use the rule $c' \to (c, out)$, then the process is iterated, and this can be done in at most n steps, because at each step we erase a prime from one occurrence of a'. If we use the rule $c' \to (c'', out)$, then in the skin membrane we have to use the rule $c'' \to (\lambda, out)$ and the string is sent out. Consequently, we have

$$L_{left}(\Pi) \cap \{a, b, b'\}^* \{d\} = \{(ab^n b')^n d \mid n \geqslant 1\}.$$

The family $ET0L$ is a full AFL, hence we can erase the symbols b' and d by a morphism and we get the language $\{(ab^n)^n \mid n \geqslant 1\}$, which is not in $ET0L$ (see [16, Vol. 1, p. 272]). Therefore, $L_{left}(\Pi) \notin ET0L$. □

Theorem 5. $0L \subset RP_2(left)$, $E0L \subseteq ERP_2(left)$.

Proof. Let $G=(V,w,P)$ be a $0L$ system. We construct the P system

$$\Pi=(V',V',[_1[_2\ ]_2]_1,\{w'c\},\emptyset,R_1,R_2),$$

$$R_1=\{a'\to x \mid a\to x\in P\}\cup\{c\to(c,in),\ c\to(\lambda,out)\},$$

$$R_2=\{a\to a' \mid a\in V\}\cup\{c\to(c,out)\},$$

where $V'=\{a' \mid a\in V\}\cup\{c\}$, and w' is the string obtained by priming all symbols of w. The equality $L_{left}(\Pi)=L(G)$ is obvious, hence we have the inclusion $0L\subseteq RP_2(left)$. The inclusion is proper, because there are finite languages which are not in $0L$, but, clearly, each finite language is in $RP_1(left)$.

If we start from an $E0L$ system $G=(V,T,w,P)$ and we construct Π as above, with the terminal alphabet T, then we obtain the inclusion $E0L\subseteq ERP_2(left)$ (for which we do not know whether it is proper or not). □

Theorem 6. *$MAT\subset ERP_4(left)$, $MAT\subseteq ERP_4(free)$.*

Proof. Because $0L\subseteq RP_2(left)$ (Theorem 5) and $0L-MAT\neq\emptyset$ ($0L$ contains one-letter non-regular languages), it follows that $RP_2(left)-MAT\neq\emptyset$, hence we only have to prove the inclusions.

To this aim, we start by considering a matrix grammar (without appearance checking) $G=(N,T,S,M)$, in the binary normal form, that is, with $N=N_1\cup N_2\cup\{S\}$, and with the matrices in M of the forms $(S\to ZB),(X\to\alpha,A\to x)$, for $Z,X\in N_1,\alpha\in N_1\cup\{\lambda\}$, $A,B\in N_2,x\in(N_2\cup T)^*$. We replace each matrix $(X\to\lambda,A\to x)$ with $(X\to f,A\to x)$, where f is a new symbol. Assume the two-rule matrices (from M or modified as above) labeled in a one-to-one manner with m_i, $1\leqslant i\leqslant k$ (note that $(S\to ZB)$ is the unique matrix of this form in M). We construct the P system

$$\Pi=(V,T,[_1[_2[_3[_4\ ]_4]_3]_2]_1,BZ,\emptyset,\emptyset,\emptyset,R_1,R_2,R_3,R_4)$$

with

$$\begin{aligned}
V=&\,N_1\cup N_2\cup T\cup\{f,f'\}\\
&\cup\{X' \mid X\in N_1\}\cup\{A' \mid A\in N_2\}\\
&\cup\{X_{i,j} \mid 1\leqslant i,j\leqslant k\}\cup\{A_{i,j} \mid 1\leqslant i,j\leqslant k\},\\
R_1=&\,\{C\to C' \mid C\in N_2\}\\
&\cup\{A\to(A_{i,1},in) \mid m_i:(X\to\alpha,A\to x),1\leqslant i\leqslant k\}\\
&\cup\{f\to(\lambda,out)\},\\
R_2=&\,\{X\to(X_{i,1},in) \mid m_i:(X\to\alpha,A\to x),1\leqslant i\leqslant k\}\\
&\cup\{X_{i,j}\to(X_{i,j+1},in) \mid X\in N_1,1\leqslant j<i\leqslant k\}\\
&\cup\{\alpha'\to(\alpha,out) \mid \alpha\in N_1\cup\{f\}\},
\end{aligned}$$

$$R_3 = \{A_{i,j} \to (A_{i,j+1}, out) \mid A \in N_2, 1 \leqslant j < i \leqslant k\}$$
$$\cup\, \{A_{i,i} \to (x, in) \mid m_i : (X \to \alpha, A \to x), 1 \leqslant i \leqslant k\}$$
$$\cup\, \{\alpha' \to (\alpha', out) \mid \alpha \in N_1 \cup \{f\}\},$$
$$R_4 = \{X_{i,i} \to (\alpha', out) \mid m_i : (X \to \alpha, A \to x), 1 \leqslant i \leqslant k\}$$
$$\cup\, \{C' \to C \mid C \in N_2\}.$$

Let us examine how Π works in the leftmost mode. Assume that we have a string of the form wX in membrane 1 (initially, we have the string BZ, for $(S \to ZB)$ being the start matrix of G). From left to right, we can prime the non-terminals of w and in any moment we can use a rule $A \to (A_{i,1}, in)$, for some $1 \leqslant i \leqslant k$, and the string is sent to membrane 2 (if all symbols are primed, then the string will remain forever unchanged). The only applicable rule in membrane 2 is $X \to (X_{j,1}, in)$, for some $1 \leqslant j \leqslant k$, and the string is sent to membrane 3. Here the second component of the subscript of A is increased by one and the string is sent to membrane 2, where we do the same with the second component of the subscript of the symbol X, and the string is sent to membrane 3 again. Note that the subscripted symbols are unique from N_1 and N_2, hence the leftmost restriction is observed. The string circulates between membranes 2 and 3 until one of the symbols gets a subscript of the form i,i or j,j. If $i < j$, then this happens for $A_{i,i}$, a string of the form $w'X_{j,i}$ is sent to membrane 4 and it gets stuck here. If we have $j < i$, then the string gets stuck in membrane 2, in the form $w'X_{j,j}$. In both cases, we have no output. If $i = j$, then the string is sent to membrane 4 by the rule $A_{i,i} \to (x, in)$ and here, after using the rules $C' \to C, C \in N_2$, for all primed non-terminals, we use the rule $X_{i,i} \to (\alpha', out)$. Because of α', the string is sent to membrane 2 and from here to membrane 1, with α non-primed. The process can be iterated as long as we do not have $\alpha = f$. If any non-terminal is still present, then we can use the rules of the form $C \to C'$ for all of them and then we can send out the string by using the rule $f \to (\lambda, out)$, or we can use the rule $A \to (A_{i,1}, in)$, which blocks the string in membrane 2. Thus, only if the string is terminal, it belongs to $L_{left}(\Pi)$. Consequently, $L_{left}(\Pi) = L(G)$ and we have the inclusion $MAT \subseteq ERP_4(left)$.

The reader can easily see that $L_{left}(\Pi) = L(\Pi)$: the rules $C \to C', C' \to C$, for $C \in N_2$, are both useless and harmless in the free mode of derivation, we generate the same language irrespective whether we use them or not. Thus, we also have the inclusion $MAT \subseteq ERP_4(free)$. (In Section 6 we will see that, in fact, this is an equality.) □

6. Permitting and forbidding conditions in P systems

We introduce now two further restrictions in the use of rules of a rewriting P system, classical in formal language theory. First, we consider P systems where the rules are applied with respect to some *forbidding conditions*. This means that the rules are of the form $\langle X \to x; F\rangle$, where $F \subseteq V$, and the rule $X \to x$ can be applied only to the strings which do not contain any symbol from F (when $F = \emptyset$ this means that the rule is applied without any restriction, in the free mode). We denote by $L_{forb}(\Pi)$ the language generated by a P system Π using such rules, and by $ERP_m(forb)$, $m \geqslant 1$, we denote the family of all such languages, generated by systems with at most m membranes. Then, we consider the application of the rules with *permitting conditions*; the rules are still of the form $\langle X \to x; F\rangle$, where $F \subseteq V$, but now the rule $X \to x$ can be applied only to the strings which contain all symbols in F (again, $F = \emptyset$ means no restriction, hence the free application of the rule). We denote by $L_{perm}(\Pi)$ the language generated by a P system Π using such rules and by $ERP_m(perm)$, $m \geqslant 1$, we denote the family of all such languages, generated by systems with at most m membranes. As previously said, if the degree of the systems is not bounded, then we replace the subscript m by $*$.

First of all, we give some preliminary results analogous to the results in Lemma 1. Directly from the definitions, we have

Lemma 7. (1) $[E]RP_m(free) \subseteq [E]RP_m(perm)$, $m \geqslant 1$;
(2) $RP_m(perm) \subseteq ERP_m(perm)$, $m \geqslant 1$;
(3) $[E]RP_m(perm) \subseteq [E]RP_{m+1}(perm)$, $m \geqslant 1$.

Lemma 8. (1) $[E]RP_m(free) \subseteq [E]RP_m(forb)$, $m \geqslant 1$;
(2) $RP_m(forb) \subseteq ERP_m(forb)$, $m \geqslant 1$;
(3) $[E]RP_m(forb) \subseteq [E]RP_{m+1}(forb)$, $m \geqslant 1$.

In the next theorem we show that, when we use forbidding conditions associated with rules, two membranes suffice to generate all RE languages, and this is true even for non-extended systems:

Theorem 9. $RE = RP_2(forb) = RP_*(forb) = ERP_2(forb) = ERP_*(forb)$.

Proof. According to the Turing–Church thesis and to Lemma 8, we only have to prove the inclusion $RE \subseteq RP_2(forb)$. Consider a matrix grammar with appearance checking $G = (N_1 \cup N_2, T, S, M, F)$ in the binary normal form. We assume the matrices labeled in a one-to-one manner with $m_1, \dots, m_{k_1}$ (matrices of type 2), $m_{k_1+1}, \dots, m_{k_2}$ (matrices of type 3), $m_{k_2+1}, \dots, m_n$ (matrices of type 4).

We build the P system

$$\Pi = (V, [_0[_1\ ]_1]_0, \{ZB\}, \emptyset, R_0, R_1),$$

where $V = N_1 \cup N_2 \cup T \cup \{f\} \cup \{X_i \mid X \in N_1,\ 1 \leqslant i \leqslant n\}$ and the rules of R_0 and R_1 are defined as follows.

In R_1 we place the rules
$\langle X_i \to (Y, out); \emptyset\rangle$, s.t. $m_i = (X \to Y, A \to x)$, $1 \leqslant i \leqslant k_1$, is a type 2 matrix
$\langle X_i \to (f, out); \emptyset\rangle$, s.t. $m_i = (X \to \lambda, A \to x)$, $k_2 + 1 \leqslant i \leqslant n$, is a type 4 matrix.

For each matrix of type 2, $m_i = (X \to Y, A \to x), 1 \leqslant i \leqslant k_1$, and of type 4, $m_i = (X \to \lambda, A \to x), k_2 + 1 \leqslant i \leqslant n$, we place in R_0 the rules
$\langle X \to X_i; \emptyset\rangle$,
$\langle A \to (x, in); \{X\} \cup \{X_j \mid j \neq i\}\rangle$.

For each matrix of type 3, $m_i = (X \to Y, A \to \#), k_1 + 1 \leqslant i \leqslant k_2$, we place in R_0 the rule

$$\langle X \to Y; \{A\}\rangle.$$

Moreover, we add to R_0 the rule $\langle f \to (\lambda, out); N_1 \cup N_2\rangle$.

Consider a string of the form Xw in membrane 0 (initially we have $X = Z$ and $w = B$). The only rules which can be applied are the rules of the form $\langle X \to X_i; \emptyset\rangle$ and $\langle X \to Y; \{A\}\rangle$, which simulate the first production of a matrix.

If we apply a rule $\langle X \to X_i; \emptyset\rangle$, where i is the label of a matrix of type 2, then we can only simulate the second production of the same matrix with the rule $\langle A \to (x, in); \{X\} \cup \{X_j \mid j \neq i\}\rangle$. In fact, the forbidding condition for the rule, $\{X\} \cup \{X_j \mid j \neq i\}$, prevents the application of the second production of a different matrix. Then, the string is sent to membrane 1 where we complete the simulation of the matrix and we send back in membrane 0 a string ready for the simulation of another string.

If we apply a rule $\langle X \to X_i; \emptyset\rangle$, where i is the label of a matrix of type 4, then the process is similar. The only difference is that the string which is sent from membrane 1 to membrane 0 is of the form fw. If w is terminal, then we can apply the rule $\langle f \to (\lambda, out); N_1 \cup N_2\rangle$, which sends out of the system a terminal string. This rule cannot be applied if w contains non-terminal symbols, due to the forbidding condition $N_1 \cup N_2$.

The simulation of a type 3 matrix is done with the rules $\langle X \to Y; \{A\}\rangle$. These rules can be applied only if the string does not contain the corresponding symbol from N_2, and, once applied, the simulation of the type 3 matrix is completed, thus the string is ready for the simulation of another matrix. It is easy to see that $L(G) = L_{forb}(\Pi)$. □

Surprisingly enough, the use of permitting conditions with ERP systems not only does not lead to a characterization of RE languages, but it simply does not increase the power of rewriting P systems: the class of languages generated by such systems coincides with the class of languages generated by matrix grammars without appearance checking.

Theorem 10. *$MAT = ERP_*(perm) = ERP_2(perm)$.*

Proof. The inclusion $ERP_2(perm) \subseteq ERP_*(perm)$ follows directly from Lemma 7.

The inclusion $MAT \subseteq ERP_2(perm)$ can be proved by using a construction very similar to the one in the previous proof. The only difference is that the rules of the form $\langle A \to (x, in); \{X\} \cup \{X_j \mid j \neq i\}\rangle$ (with forbidding conditions) are replaced by rules $\langle A \to (x, in); \{X_i\}\rangle$ (with permitting conditions): the second production of the matrix i is simulated only if the string contains the symbol X_i.

To prove the inclusion $ERP_*(perm) \subseteq MAT$, let us consider an ERP system with permitting conditions, $\Pi = (V, T, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n)$ generating the language $L_{perm}(\Pi)$. We consider the skin membrane labeled with 1.

Clearly, each string present at any time in the system is rewritten independently with respect to the other strings present in the system at the same time. The only possible mutual influence is the fact that if a string can be rewritten forever, then no output of the computation is accepted. Thus, if $L_{perm}(\Pi) \neq \emptyset$ (otherwise, the language is trivially in MAT), then there is an axiom w which leads to a string in T^* which exits the system, while other axioms either lead to strings which leave the system or to strings to which no rule can be applied. Thus, the axioms and their descendant strings can be assumed to evolve independently to each other, without taking care whether or not other strings can be rewritten forever.

We build a matrix grammar (without appearance checking) $G=(N,T,S,M)$, generating the same language as Π in the following way (h is the morphism defined by $h(a)=a', a\in V$):

$$
\begin{aligned}
N &= \{a' \mid a \in V\} \cup \{[i] \mid 1 \leqslant i \leqslant n\} \cup \{E,S\},\\
M &= \{(S \to [i]h(w)) \mid w \in M_i, 1 \leqslant i \leqslant n\}\\
&\quad \cup \{([i] \to [i], a_1' \to a_1', \ldots, a_k' \to a_k', A' \to h(x)) \mid\\
&\qquad \langle A \to (x, here); \{a_1,\ldots,a_k\}\rangle \in R_i, 1 \leqslant i \leqslant n\}\\
&\quad \cup \{([i] \to [j], a_1' \to a_1', \ldots, a_k' \to a_k', A' \to h(x)) \mid\\
&\qquad \langle A \to (x, in); \{a_1,\ldots,a_k\}\rangle \in R_i, 1 \leqslant i \leqslant n, \text{ and } j \text{ is the label}\\
&\qquad \text{of a membrane placed directly inside membrane } i\}\\
&\quad \cup \{([i] \to [j], a_1' \to a_1', \ldots, a_k' \to a_k', A' \to h(x)) \mid\\
&\qquad \langle A \to (x, out); \{a_1,\ldots,a_k\}\rangle \in R_i, 2 \leqslant i \leqslant n, \text{ and } j \text{ is the label}\\
&\qquad \text{of the membrane surrounding membrane } i\}\\
&\quad \cup \{([1] \to E, a_1' \to a_1', \ldots, a_k' \to a_k', A' \to h(x)) \mid\\
&\qquad \langle A \to (x, out); \{a_1,\ldots,a_k\}\rangle \in R_1\}\\
&\quad \cup \{(E \to E, a' \to a) \mid a \in T\}\\
&\quad \cup \{(E \to \lambda)\}.
\end{aligned}
$$

It is easy to see that the symbols $[i], 1\leqslant i\leqslant n$, control the work of the grammar G in such a way that the computations in Π are correctly simulated: the symbols $[i]$ indicate the rules to be used as well as the membrane where the corresponding string is placed in the next configuration. At the same time, the mode of working specific to matrix grammars, makes possible the checking of the permitting conditions by using rules of the form $a' \to a'$, for a being an element of a permitting conditional set of a rule. In this way, all computations in Π can be simulated by derivations in G, working with primed symbols. When a string is to be sent out of the system, the symbol E is introduced, no further rule from Π can be simulated, and the symbols of T lose their primes. If no symbol from V–T is present and if the unpriming is complete, then we get a string in $L(G)$. Consequently, $L_{perm}(\Pi) = L(G)$, which completes the proof. □

Combining the results in Theorems 6 and 10, we obtain the following unexpected result:

Corollary 11. $MAT = ERP_4(free) = ERP_*(free) = ERP_2(perm) = ERP_*(perm)$.

7. Final remarks

We have investigated P systems with string objects processed by rewriting rules, in particular, by considering some restrictions in the use of the rules which are classic in formal language theory: leftmost derivation and conditional use of rules. When using extended rewriting P systems with leftmost derivations, four membranes suffice to obtain universality. The universality can be obtained using non-extended rewriting P systems with forbidding conditions, with only two membranes. Surprisingly enough, we cannot obtain the same result when using permitting conditions, even if we use extended rewriting P systems with an unbounded number of membranes.

Several problems still remain open, mainly concerning the optimality of the number of membranes used in these results.

Acknowledgements

We are very much indebted to two anonymous referees, who have carefully read the initial version of the paper. In particular, the present form of Theorem 3, improving the original result, has been suggested by one of the referees. We thank him very much.

References

[1] P. Bottoni, A. Labella, C. Martin-Vide, Gh. Păun, Rewriting P systems with conditional communication, in: W. Brauer, H. Ehrig, J. Karhumaki, A. Salomaa (Eds.), Formal and Natural Computing, Essays Dedicated to Gregorz Rozenberg, Lecture Notes in Computer Science, 2300, Springer, Berlin, 2002, pp. 325–353.

[2] C. Calude, Gh. Păun, Computing with Cells and Atoms, Taylor & Francis, London, 2000.

[3] J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, IEEE Seventh Internat. Conf. on String Processing and Information Retrieval, SPIRE 2000, La Coruna, Spain, pp. 64–74.

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

[5] R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes: three more collapsing hierarchies, 2000, manuscript.

[6] R. Freund, Gh. Păun, On the number of variables in graph-grammars, programmed, and matrix grammars, Proc. MCU Conf., Chişinău, Lecture Notes in Computer Science, vol. 2055, Springer, Berlin, 2001, pp. 214–225.

[7] V. Geffert, Context-free-like forms for the phrase-structure grammars, Proc. MFCS'88, Lecture Notes in Computer Science, vol. 324, Springer, Berlin, 1988, pp. 309–317.

[8] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, Acta Inform. 31 (1994) 719–728.

[9] S.N. Krishna, R. Rama, On the power of P systems with sequential and parallel rewriting, Internat J. Comput. Math. 77 (1–2) (2000) 1–14.

[10] S.N. Krishna, R. Rama, P systems with replicated rewriting, J. Automata Languages Combin. 6 (3) (2001) 345–350.
[11] V. Manca, C. Martin-Vide, Gh. Păun, On the power of P systems with replicated rewriting, J. Automata Languages Combin. 6 (3) (2001) 359–374.
[12] C. Martin-Vide, Gh. Păun, Computing with membranes, One more collapsing hierarchy, Bull. EATCS 72 (2000) 183–187.
[13] C. Martin-Vide, Gh. Păun, String objects in P systems, Proc. Workshop on Algebraic Systems, Formal Languages and Computations, Kyoto, 2000, RIMS Kokyuroku, Kyoto Univ., 2000, pp. 161–169.
[14] Gh. Păun, Computing with membranes, J. Comput. System Sci. 61 (1) (2000) 108–143 (see also Turku Center for Computer Science-TUCS Report No. 208, 1998, www.tucs.fi).
[15] G. Rozenberg, A. Salomaa, The Mathematical Theory of L Systems, Academic Press, New York, 1980.
[16] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Heidelberg, 1997.
[17] Cl. Zandron, G. Mauri, Cl. Ferretti, Universality and normal forms on membrane systems, in: R. Freund, A. Kelemenova (Eds.), Proc. Internat. Workshop Grammar Systems 2000 Bad Ischl, Austria, July 2000, pp. 61–74.