Universality and normal forms on membrane systems

C. Zandron, G. Mauri, C. Ferretti

Dipartimento di Informatica Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20136 Milano
Università degli Studi di Milano-Bicocca - ITALY

contact: ferretti@disco.unimib.it

Abstract

The work presents techniques and details for some improvements and new results concerning membrane systems. These systems can be defined following several variations in their formalisms, and every model starts a race to the simplest possible universal set of features. We try to outline our results on universality and previous ones in a common framework, following our understanding of the essentials shared by all the techniques based on simulation between grammar systems.

Moreover, we contrast that approach to further results we obtained with techniques based on normal forms, which we define on some membrane systems.

1 Introduction

P systems were recently introduced in [1] as a class of distributed parallel computing devices of a biochemical type.

The basic model consists of a membrane structure composed by several cell-membranes, hierarchically embedded in a main membrane called the skin membrane. The membranes delimit regions and can contain objects. The objects evolve according to given evolution rules associated with the regions. A rule can modify the objects and send them outside the membrane or to an inner membrane. Moreover, the membranes can be dissolved. When a membrane is dissolved, all the objects in this membrane remain free in the membrane placed immediately outside, while the evolution rules of the dissolved membrane are lost. The skin membrane is never dissolved.

The evolution rules are applied in a maximally parallel manner: at each step, all the objects which can evolve should evolve. A computation device is obtained: we start from an initial configuration and we let the system evolve. A computation halts when no further rule can be applied. The set of objects in a specified output membrane are the result of the computation.

Many variants are considered in [1],[5], [6], [7] and [9].

We consider here the variant introduced in [5] as Rewriting Super-Cell system (or RP system), in which the objects can be described by finite strings over a given finite alphabet, instead of objects of an atomic type (i.e. elements of a finite alphabet). The evolution of an object will correspond to a transformation of the string: the evolution rules are given as context-free rewriting rules.

It is known that a system of this type with three membranes and with priority relation among evolution rules (that is, a rule can be applied only when no other rule of higher priority can be applied) is able to generate every Recursively Enumerable language ([5]). We will show that priority relations over evolution rules are even more powerful: in fact, two membranes are enough to characterize the Recursively Enumerable languages.

In [6] a new feature is introduced: the membranes can be made thicker or thinner (and also dissolved as said before). Initially, all membranes have thickness 1. If a membrane has thickness 2, then no object can pass through it. In [6] it was shown that P systems with this feature characterize the one-letter Recursively Enumerable languages, and one conjecture was made: a characterization of Recursively Enumerable languages can be obtained with RP systems using context-free rewriting rules without priority relations. We will show that the conjecture was correct: if we use this new feature, we can characterize the Recursively Enumerable languages with a RP system that does not use priorities. Other results are given for RP systems with a fixed number of membranes and which do not use priorities.

Another variant proposed in [6] is about the communication of objects to inner membranes: the evolution rules do not specify the precise membrane where the objects will be sent after the application of the rule. Instead, "electrical charges" are used, associated to membranes and to objects: they can be marked with "positive" (+), "negative" (-) or "neutral". An object marked with + (respectively -) will enter a membrane marked with - (respectively +), nondeterministically chosen from the set of membranes immediately inside the region where the object is produced. The neutral objects are not moved to an inner membrane.

As stated in [6], this feature is useful to obtain more "realistic" systems (with biochemical features instead than artificial ones). We show here that it is also possible to get simple systems. We give two normal forms for RP systems with this feature. First, we'll show that every language generated by a RP system with priority and without dissolving action can be generated by an equivalent system with two rules in each membrane and with a simple structure (the skin membrane contains elementary cells only). The second normal form is about RP systems without priority: we'll show that every systems of that type (under certain restrictions) is equivalent to a system of the same type with three rules in each membrane.

An interesting approach to normal forms for P-system can also be found in [11].

2 Rewriting P systems

We do not give a complete formal definition of (rewriting) P system; details and examples can be found in [4] and [5]. We refer to [11] for elements of Formal Language Theory.

A membrane *structure* is a construct consisting of several membranes placed in a unique membrane; this unique membrane is called *skin* membrane. We identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane.

A membrane identifies a region, delimited by it and by the membranes immediately inside it. If in the regions we place multisets of objects from a specified finite set V, we get a super-cell.

A super-cell system (or P system) is a super-cell provided with evolution rules for its objects and with a designated output membrane.

We consider instead rewriting super-cell systems (RP systems). In such systems objects can be described by finite strings over a given finite alphabet. The evolution of an object will correspond to a transformation of the string. Consequently, the evolution rules are given as rewriting rules. The rules are also provided with indications of the target membrane of the produced string, and they can contain a symbol that describes the action of the rule on the membrane: the thickness of the membrane can be decreased (using the symbol δ) or increased (using the symbol τ). We only use context-free rules.

Such a system of degree $n, n \geq 1$, is a construct

$$\Pi = (V, \mu, M_1, \ldots, M_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0)$$

where: • V is an alphabet

• μ is a membrane structure consisting of n membranes (labeled with 1, ..., n)

• $M_i, 1 \leq i \leq n$ are finite languages over V

• $R_i, 1 \leq i \leq n$ are finite sets of context free evolution rules. They are of the form $X \rightarrow v(tar)$ where X is a symbol of V and $v = v^{'}$ or $v = v^{'}\delta$ or $v = v^{'}\tau$. $v^{'}$ is a string over V and δ, τ are special symbols not in V. $tar \in \{here, out, in_m\}, 1 \leq m \leq n$ represents the *target* membrane, i.e. the membrane where the string produced with this rule will go. Often, the indication "here" is omitted.

• $\rho_i, 1 \leq i \leq n$ are partial order relations over R_i

• i_0 is the output membrane.

The membrane structure μ and the finite languages $M_1, ..., M_n$ constitute the initial configuration of the system. In the initial configuration all membranes are considered of thickness 1. We can pass from a configuration to another by applying in parallel the evolution rules to all strings which can be rewritten, obeying the priority relations. The strings generated are collected in a designated membrane, the output one.

Note that each string is processed by one rule only; the parallelism refers to act of processing simultaneously all available strings by all applicable rules. If several rules can be applied to a string, then we take only one rule and only one possibility to apply it, and we consider the obtained string as the next state of the object described by the string. It is important to underline the fact that the evolution of strings is not independent, but interrelated in two ways:

1) if we have priorities, a rule r_1 applicable to a string x can forbid the use of another rule, r_2, of lower priority for rewriting another string y which is present at the same time in the same membrane. Then, we can apply r_2 on y only if r_1 is not applicable to it nor to any string $x^{'}$ obtained from x by using r_1.

2) Even without priority, if a string can be rewritten forever, in a membrane or on an itinerary through several membranes, and this cannot be avoided, then all strings are lost, because the computation never stops.

Moreover, a rule specifies, as previously said, the target membrane. If a rule contains "here" as target membrane (or the target is not specified), it means that the string obtained after the rule is applied will remain in the same region where the rule is applied. If the specified target is "out" and the thickness of the membrane is 1, the string will be sent to the region placed immediately outside. Finally, if the specified target is "in_m", where m is a label of a membrane adjacent to the region of the rule, and the membrane m has thickness 1, then the string is sent to that membrane.

If a rule contains the special symbol δ and the membrane where this rule is applied has thickness 1, then that membrane is dissolved and it is no longer recreated; the objects in the membrane become objects of the membrane placed immediately outside, while the rules of the dissolved membrane are removed. If the membrane has thickness 2, this symbol reduces the thickness to 1. If a rule contains the special symbol τ the thickness of the membrane where this rule is applied is increased; the thickness of a membrane of thickness 2 is not further increased. If both these symbols are introduced in the same region, the corresponding membrane preserves its thickness.

The transfers of objects have priority over the actions of δ and τ: if at the same step an object has to pass through a membrane and a rule changes the thickness of that membrane, then we first transmit the object, and after that we change the thickness.

We denote by $L(\Pi)$ the language generated by Π and by $RP(Pri, \delta, \tau)$ the family of languages generated by rewriting P systems using priority and both actions indicated by δ and τ. When one of the features $\alpha \in \{Pri, \delta, \tau\}$ is not used, we write $n\alpha$ instead of α. If only systems with at most n membranes are used, then we add the subscript n to RP.

In the last section we will consider systems with rules R_i specifying the target in a different way: • $R_i, 1 \leq i \leq n$ are finite sets of context free evolution rules. They are of the form $X \rightarrow v(p)$ where X is a symbol of V and $v = v^{'}$ or $v = v^{'}\delta$. $v^{'}$ is a string over V, δ is a special symbol not in V and $p \in \{here, out, +, -\}$. Often, the indication "here" is omitted.

Moreover, a rule can mark the object with +, -, out, here. If a rule marks a string with "here" (or if the mark is omitted), it means that the string obtained

after the rule is applied will remains in the same region where the rule is applied. If the mark is "out" the string will be send to the region placed immediately outside. If the string is marked with + (or -), it will be sent through a membrane marked with - (respectively +) and adjacent to the region where the rule is applied.

3 The power of rewriting P systems

We show now that rewriting P systems are very powerful. If we use a partial order relation between the rules, we are able to generate any recursively enumerable language with a very simple model: two membranes are enough. This sharpen a result in [5], where this result is obtained by using three membranes.

In this and in the other proofs of the paper we use the notion of matrix grammar. Such a grammar is a construct $G = (N, T, S, M, C)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, ..., A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and C 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 $(A_1 \to x_1, ..., A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, 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 $w_i = w_{i+1}, A_i$ does not appear in w_i, and the rule $A_i \to x_i$ appears in C (the rules of a matrix are applied in order, possibly skipping the rules in C if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $C = \emptyset$ then the grammar is said to be without appearance checking (and C is no longer mentioned). We denote by $\Rightarrow^*$ the reflexive and transitive closure of the relation $\Rightarrow$. The language generated by G is defined by $L(G) = \{w \in T^* | S \Rightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT.

A matrix grammar $G = (N, T, S, M, C)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets mutually disjoint, and the matrices in M are of 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. $(X \to Y, A \to \dagger)$, 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 C consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3. One sees that $\dagger$ is a trap-symbol: once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation (clearly, matrices of forms 2 and 3 cannot be used at the last step of a derivation). According to Lemma 1.3.7 in [5], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

We denote by CF and RE the families of context-free and recursively enumerable languages respectively. It is known that $CF \subset MAT \subset MAT_{ac} = RE$. Further details about matrix grammars can be found in [2] and in [11]. Moreover, in [3] it is shown that the one-letter languages in MAT are regular.

Theorem 1. $RE = RP_2(Pri, n\delta, n\tau)$

Proof. (Sketch) The inclusion $RP_2(Pri, n\delta, n\tau) \subseteq RE$ follows directly from the Church-Turing thesis. We prove here the opposite inclusion. Consider a matrix grammar with appearance checking $G = (N, T, S, P, F)$ in the normal form previously described. Then consider a grammar G' in which we substitute each matrix of type 4 $(X \to x_1, A \to x_2)$, with $x_1, x_2 \in T^*$, with a matrix $(X \to X'x_1, A \to x_2)$ (which is considered of type 4'), and with the additional matrices $(X' \to \lambda)$, where X' is a symbol not in G associated with X. Clearly, $L(G) = L(G')$. We assume the matrices of the types 2, 3 and 4' labeled in a one-to-one manner, with $m_1, m_2, ..., m_k$. We show how to construct a Rewriting Super Cell system with 2 membranes that generates the same language of G'. :

$$\Pi = (V, \mu, M_1, M_2, (R_1, \rho_1), (R_2, \rho_2), 2)$$

$$V = N_1 \cup N_2 \cup \{E, Z, \dagger\} \cup T \cup \{X_i, X_i^{'} | X \in N_1, 1 \leq i \leq k\}$$

$$\mu = [_1[_2]_2]_1$$

$$M_1 = \{XAE | S \to XA \ from \ matrix \ of \ type \ 1\}, \text{while } M_2 \text{ is empty}$$

$$\begin{aligned}
R_1 = \ & \{r_\alpha : \alpha \to \alpha | \alpha \in V - T, \alpha \neq E\} \\
\cup \ & \{r_0 : E \to \lambda(in_2)\} \\
\cup \ & \{\dagger \to \dagger\} \\
\cup \ & \{X \to Y_i(in_2) | \\
& m_i : (X \to Y, A \to z_i) \ is \ a \ matrix \ of \ type \ 2 \ or \ 3, \\
& z_i \in (N_2 \cup T)^* or z_i = \dagger, 1 \leq i \leq k\} \\
\cup \ & \{X \to X_i^{'} x_1(in_2), X_i^{'} \to \lambda | \\
& m_i : (X \to X^{'} x_1, A \to x_2) \ is \ a \ matrix \ of \ type \ 4'\} \\
\cup \ & \{Y_i \to Y, Y_i^{'} \to Y | \\
& Y \in N_1, 1 \leq i \leq k\}
\end{aligned}$$

$$\rho_1 = \{r_\alpha > r_0 | \alpha \in V - T, \alpha \neq E\}$$

$$
\begin{aligned}
R_2 \;=\;& \{r_i : Y_i \to Y_i, r_i^{'} : A \to x(out) | \\
& m_i : (X \to Y, A \to x) \; is \; a \; matrix \; of \; type \; 2\} \\
\cup\;& \{p_i : Y_i \to Y_i^{'}, p_i^{'} : Y_i^{'} \to Y_i, \\
& p_i^{''} : A \to \dagger(out) | \\
& m_i : (X \to Y, A \to \dagger) \; is \; a \; matrix \; of \; type \; 3\} \\
\cup\;& \{q_i : X_i^{'} \to X_i^{'}, q_i^{'} : A \to x_2 | \\
& m_i : (X \to X^{'} x_1, A \to x_2) \; is \; a \; matrix \; of \; type \; 4'\} \\
\cup\;& \{s_0 : E \to E(out)\}
\end{aligned}
$$

$$
\begin{aligned}
\rho_2 \;=\;& \{r_i > r_j^{'} | i \neq j, for \; all \; i, j\} \\
\cup\;& \{r_i > p_j^{''} | for \; all \; i, j\} \\
\cup\;& \{r_i > q_j^{'} | for \; all \; i, j\} \\
\cup\;& \{r_i > s_0 | for \; every \; i\} \\
\cup\;& \{p_i^{''} > p_i > s_0 | for \; every \; i\} \\
\cup\;& \{p_i > r_j^{'} | for \; all \; i, j\} \\
\cup\;& \{p_i > p_j^{''} | i \neq j, for \; all \; i, j\} \\
\cup\;& \{p_i > q_j^{'} | for \; all \; i, j\} \\
\cup\;& \{p_i^{'} > r_j^{'} | for \; all \; i, j\} \\
\cup\;& \{p_i^{'} > p_j^{''} | i \neq j, for \; all \; i, j\} \\
\cup\;& \{p_i^{'} > q_j^{'} | for \; all \; i, j\} \\
\cup\;& \{q_i > r_j^{'} | for \; all \; i, j\} \\
\cup\;& \{q_i > p_j^{''} | for \; all \; i, j\} \\
\cup\;& \{q_i > q_j^{'} | i \neq j, for \; all \; i, j\} \\
\cup\;& \{q_i > s_0 | \; for \; every \; i\}
\end{aligned}
$$

If we simulate the first production of a type 2 matrix, we send the obtained string in membrane 2 where we can use the rule $Y_i \to Y_i$ until we apply the rule $A \to m(out)$, that simulates the production in the second position of the type 2 matrix labeled with i and send the string in membrane 1. If the symbol A is not in the string, the computation will never halt.

If we simulate the first production of a type 4' matrix, we send the obtained string in membrane 2 where we can use the rule $X_j^{'} \to X_j^{'}$ until we apply the rule $A \to x_2(out)$, that simulates the production in the second position of the type 4' matrix labeled with j and send the string in membrane 1. If the symbol A is not in the string, the computation will never halts.

Because of the priority relations, if the symbol A is in the string we have to apply the rule $p_k^{''} : A \to \dagger(out)$. The computation will never halts (in membrane 1 we have the rule $\dagger \to \dagger$).

If the symbol A is not in the string, we can apply the rule $Y_j \to Y_j^{'}$, and we get the string $Y_j^{'}wE$. On this string, we can apply a rule $Y_j^{'} \to Y_j$, and we get again the string Y_jwE, or a rule $E \to E(out)$, to send the string in membrane 1.

Summarizing, if the symbol A is in the string we introduce the trap symbol $\dagger$, otherwise we have to apply the rules $Y_j \to Y_j^{'}$ and $Y_j^{'} \to Y_j$ until we apply $E \to E(out)$ that sends back in membrane 1 the string $Y_j^{'}wE$.

In membrane 1 we can replace the symbols Y_i and $Y_i^{'}$ with Y and the symbol $X_i^{'}$ with λ. In this way, we can iterate the process of simulating the matrices of type 2, 3 and 4'. The rule $E \to \lambda(in_2)$ is used to send a terminal string in the output membrane. If any nonterminal symbol is present in the string, this rule cannot be applied, because we have to apply a rule $r_\alpha : \alpha \to \alpha$ due to the priority relations.

In the output membrane, we collect the terminal strings generated by G', thus $L(G') = L(\Pi)$. □

Now, we denote by ORD the family of languages generated by context-free ordered grammars (i.e. context-free grammars with a partial order relation on the set of rules; a rule can be applied only when no rule of a higher priority can be used).

It is known that $CF \subset ORD \subset RE$. Obviously, we have $RP_1(Pri,\ n\delta,\ n\tau) = ORD$. We just show that, if we use priority, the introduction of one more single membrane is enough to get a system able to generate any RE language, thus a computationally complete system.

Nevertheless, in [6] it is pointed out that the priority relation among evolution rules is a feature of a formal language inspiration, which looks far from biochemistry. Consequently, we want to study the generative power of RP systems which do not use the priority feature.

We show now that RP systems using the operations δ and τ are able to generate any RE language even without the use of priority relations on the rules.

Theorem 2. $RP(nPri, \delta, \tau) = RE$

Proof. (Sketch) The inclusion $RP(nPri, \delta, \tau) \subseteq RE$ follows directly from the Church-Turing thesis. We prove here the opposite inclusion. Consider a matrix grammar with appearance checking $G = (N, T, S, P, F)$ in the normal form previously described. We assume the matrices of the types 2, 3 and 4 labeled in a one-to-one manner; we label with $m_1, ..., m_{k'}$ the matrices of type 2, with $m_{k'+1}, \ldots, m_{k''}$ the matrices of type 3 and with $m_{k''+1}, \ldots, m_k$ the matrices of type 4 ($0 \leq k^{'} < k, 1 \leq k^{''} \leq k$).

We show how to construct a Rewriting P System (of degree $k + 2$) without priority but with variable thickness that generates the same language of G.

$$\Pi = (V, \mu, M_1, M_2, \ldots, M_{k+1}, M_{k+2}, R_1, R_2, \ldots, R_{k+1}, R_{k+2}, M_{k+2})$$

$$V = N_1 \cup N_2 \cup \{E, \dagger, F, F^{'}, F_2, F_3, F_4\} \cup T \cup \{X^{'}, X_2, X_3 | X \in N_1\}$$

$$\mu = [_{k+2}[_{k+1}[_1]_1[_2]_2] \cdots [_k]_k]_{k+1}]_{k+2}$$

$$M_{k+1} = \{XAE | S \to XA \; is \; the \; rule \; of \; the \; matrix \; of \; type \; 1\} \cup \{F\}$$

$$M_{k+2} = \emptyset$$

$$M_h = \dagger, for \; 1 \leq h \leq k$$

$$\begin{aligned} R_i(1 \leq i \leq k') &= \{A \to x\delta(out) | m_i : (X \to Y, A \to x) \; type \; 2, x \in (N_2 \cup T)^*\} \\ &\cup \{F^{'} \to F\tau(out)\} \end{aligned}$$

$$\begin{aligned} R_j(k' < j \leq k^{''}) &= \{Y^{'} \to Y_2\tau\} \\ &\cup \{F^{'} \to F_2\tau\} \\ &\cup \{Y_2 \to Y_3\} \\ &\cup \{F_2 \to F_3\tau\} \\ &\cup \{A \to \dagger | m_j : (X \to Y, A \to \dagger) \; matrix \; of \; type \; 3\} \\ &\cup \{F_3 \to F_4\delta\} \\ &\cup \{Y_3 \to Y(out)\} \\ &\cup \{F_4 \to F(out)\} \end{aligned}$$

$$\begin{aligned} R_h(k^{''} < h \leq k) &= \{A \to x_2\tau(out) | \\ &\qquad m_h : (X \to x_1, A \to x_2) \; of \; type \; 4, x_1, x_2 \in T^*\} \\ &\cup \{F^{'} \to F\tau(out)\} \end{aligned}$$

$$\begin{aligned} R_{k+1} &= \{X \to Y(in_w) | m_w : (X \to Y, A \to x) \; matrix \; of \; type \; 2, x \in (N_2 \cup T)^*\} \\ &\cup \{X \to Y^{'}(in_w) | m_w : (X \to Y, A \to \dagger) \; of \; type \; 3\} \\ &\cup \{X \to x_1(in_w) | m_w : (X \to x_1, A \to x_2) \; of \; type \; 4, x_1, x_2 \in T^*\} \\ &\cup \{E \to \lambda(out)\} \\ &\cup \{F \to \lambda\} \\ &\cup \{F \to F^{'}(in_r), 1 \leq r \leq k\} \\ &\cup \{\dagger \to \dagger\} \end{aligned}$$

$$R_{k+2} = \{\alpha \to \alpha | \alpha \in V - T\}$$

It's easy to see that after a while all the membranes with a label between 1 and k become of thickness 2. So, if we still have not applied the rule $F \to \lambda$, we have to apply it. The string F disappears and the computation stops.

In the output membrane we get exactly the strings of terminal symbols generated by the grammar G, that is $L(G) = L(\pi)$. □

Informally, the key concept in the previous proof is the collaboration between the string XwE and the string F. The computation can correctly terminate only when the two strings follow the same path between the membranes' structure. If both strings use the same membrane at the same time the computation can proceed. Otherwise a membrane will be dissolved and let a string † reach the control membrane where the computation will proceed forever.

For the same reason, the deletion of the string F cannot be done before sending the other string to the output membrane.

4 Normal forms

We include in this section the statements of two results concerning normal forms of RP system because we like to link them to the previous universality results: the first normal form here is obtained in an indirect non-constructive way using those results.

Definition A RP system is in *double_2_normal_form* if it is of depth 2 and in each membrane we have 2 rewriting rules.

Theorem 3. Every RE language can be generated by a system $RP^{\pm}(Pri, n\delta, n\tau)$ in double_2_normal_form.

The system in the proof (here omitted) takes advantage from the polarization of the membranes and of the strings. The polarized strings are sent to membranes with an opposite charge (chosen in a nondeterministically way). This feature permits to control the communication of the string with only two general rules (in the skin membrane), because we do not have to specify the precise membrane where the string has to go. With one rule we simulate a matrix and with the other we stop the simulation and start the phase in which we control if the string is a terminal one. We can control if the string reaches a "wrong" membrane using the rules found in itself, as we showed in the proof.

We introduce now a normal form which can be contructively obtained from a given RP system without priority and with these additional features ([9]):

• We do not collect the string in an output membrane. Instead, we consider all the terminal strings which are sent out of the system at any time during the computation.

• If a string leaves the system but it is not terminal, then it is ignored; if a string remains in the system then it does not contribute to the language generated, even if it is terminal.

• We do not consider halting computations: we leave the process continue forever and we observe the terminal strings which leaves the system; the language consists of all these strings.

Definition A RP system is in *3_normal_form* if every membrane has 3 rewriting rules.

Theorem 4. Every language $L \in RP^{\pm}(nPri, n\delta, n\tau)$ can be generated by a RP system of the same type in 3_normal_form.

5 Conclusions

We have investigated the power of Rewriting P systems. These systems seem very powerful when we use priority relations among the rules: two membranes are enough to generate any RE language. Nevertheless, as pointed out in [6], this feature looks far from biochemistry.

Thus, we investigated the power of RP systems that don't use priority. If we consider systems in which we can modify the thickness of the membranes, we are able to generate RE languages even without using priority relations on the rules.

The P systems proposed in [6] have other "natural" features. The membranes do not have a label, so the rules do not have the possibility to directly send the objects to a specific membrane. Membranes and objects have an "electrical charge" and the objects are passed to membranes according to their charge; the charge of an object can be changed with the rules. Another difference is introduced in [7]: there is no output membrane; the output is constituted by the objects that leave the super-cell through its skin membrane.

We believe that the systems presented in this paper can be quite easily modified to get the same generative power even when using these features.

Many problems remain to be investigated. For instance, it is not known if we can generate any RE language without using priority with a RP system of variable thickness with a fixed number of membranes.

Another question is related to the synchronization in the previous systems. As pointed out in the last theorem above, the interactions among the strings is a key concept in the construction of such a system. What happens if we let the system evolve without the restriction that the rules have to be applied in parallel on all the strings? Are unsynchronized systems weaker than the synchronized ones ?

Moreover, we can think of using the rules in parallel in a different way, by applying the rules to each symbol of the string that can be rewritten, as in Lyndenmayer systems.

Finally, as pointed out in [1] and [4] the theory of computing with membranes lacks basic tools (necessary conditions and normal form theorems) for producing examples and counterexamples.

We make a little step in this direction by presenting in this paper two normal forms about Rewriting P-systems. The goal is to obtain simple systems, where it is easy to analyze, to understand their generative power and their equivalence

with other generative mechanisms. For instance, we showed that every RP system of depth two and with two rules in every membrane can generate every RE language, by showing that such a system is equivalent to a Matrix Grammar System with Appearance Checking. We do not know if RP systems without priority are able to generate RE or not. We hope that the normal form given here can be useful to give an answer to the previous open problem.

References

[1] J. Dassow, Gh. Paun, On the power of membrane computing, J. Univ. Computer Sci., 5, 2, 33–49, 1999.

[2] J. Dassow, Gh. Paun, Regulated Rewriting in Formal Language Theory, Springer-Verlag, Berlin, 1989.

[3] D. Hauschildt, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, Acta Informatica, 31 719–728, 1994.

[4] Gh Paun, Computing with membranes. An introduction, Bulletin of the EATCS, 67, 139–152, 1999.

[5] Gh. Paun, Computing with membranes, submitted, 1998. (see also TUCS Research Report No 208, November 1998 http://www.tucs.fi).

[6] Gh. Paun, Computing with membranes. A variant, *J. of Foundations of Comp.Science*, 11, 167–, 2000.

[7] Gh. Paun, G.. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Inform.*, 41, 3, 2000.

[8] Gh. Paun, T. Yokomori, Membrane computing based on splicing, proc. of 5th DIMACS Workshop on DNA Based Computers, 213–227, 1999.

[9] Gh. Paun, S. Yu, On synchronization in P systems, *Fundamenta Inform.*, 38, 4, 397–410, 1999.

[10] I. Petre, A normal form for P systems, Bulletin of EATCS, 67, 165–172, 1999.

[11] G. Rozenberg, A. Salomaa, eds., Handbook of Formal Languages, Springer-Verlag, Heidelberg, 1997.