Two Normal Forms for Rewriting P Systems*

C. Zandron, C. Ferretti, G. Mauri

DISCO - Universitá di Milano-Bicocca - Italy
E-mail: *zandron/ferretti/mauri@disco.unimib.it*

Abstract. P systems have been recently introduced by Gh. Păun as a new model for molecular computations based on the notion of membrane structure. In this work, we consider Rewriting P Systems which use electrical charges to move objects between the membranes. We show that electrical charges, a feature introduced in order to obtain more "realistic" systems, induces simple systems: we define two normal forms for two variants of Rewriting P Systems which make use of electrical charges.

Keywords: Formal languages, distributed computing, molecular computation

1 Introduction

The 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 objects in a specified output membrane are the result of the computation.

Many variants are considered in [1], [5], [6], [7] and [8].

We consider here the systems introduced in [5] as Rewriting Super-Cell systems (or RP systems), 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

* This work has been supported by the Italian Ministry of University (MURST), under project "Unconventional Computational Models: Syntactic and Combinatorial Methods".

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. In a variant proposed in [6] the evolution rules do not specify the label of the membrane where the objects are sent. 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 to the membrane delimiting the region where the object is produced. The neutral objects are not introduced into 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 that this feature induces simple systems: we give two normal forms for RP systems using this feature. First, we will 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 will 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.

A normal form for P-system can be found in [10].

2 Rewriting P Systems

A membrane structure is a construct consisting of several membranes placed in a unique membrane; this unique membrane is called a 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 (the membranes are said to be *adjacent* to the delimited region). If we place multisets of objects in the regions 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. The *depth* of a P system is equal to the height of the tree describing its membrane structure.

We consider here *Rewriting Super-Cell Systems (RP Systems) with polarized membranes.* 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. We only use context-free rewriting rules. The rules in a region can mark the objects to let them pass through the membranes as described in the following.

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

$$\Pi = (V, \mu, M_1, \ldots, M_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0), \text{ where:}$$

- V is an alphabet
- μ is a membrane structure consisting of n membranes (labeled with $1, ..., n$); each membrane is marked with one of the symbols $+$, $-$, 0.
- $M_i, 1 \leq i \leq n$ 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 $X \rightarrow v(p)$ 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. Note that the substring (p) is not inserted into the sentential form as subword. It is only used to communicate the objects through the membranes.
- $\rho_i, 1 \leq i \leq n$ 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. We can pass from a configuration to another one by using the evolution rules in parallel on all strings which can be rewritten, obeying the priority relations. Note that each string is processed by one rule only; the parallelism refers to processing simultaneously all available string 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 consider the obtained string as the next state of the object described by the string.

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 remain in the *same region* where the rule is applied. If the mark is *out*, the string will be sent to the region placed immediately *outside*. If the string is marked with $+$ (or $-$), it will be sent through an *inner* membrane marked with $-$ (respectively $+$) and adjacent to the region where the rule is applied. If no such a membrane exists, then the rule cannot be applied.

If a rule contains the special symbol δ then the membrane where the rule is applied 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.

A sequence of transitions between configurations of a P system Π, is called a *computation* with respect to Π. A computation *halts* when there is no rule applicable to the objects present in the last configuration. The strings collected in the output membrane constitute the *output* of the P system. If a computation never stops, then it provides no output.

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, 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 and to the string $x^{'}$ obtained from x by using r_1.
2. Even without priority, if a string can be rewritten forever then all strings are lost, because the computation never stops.

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

More details and examples about P systems can be found in [4], [5] and [6]. For elements of Formal Language Theory, we refer to [11].

3 Rewriting P Systems with Priority and Polarized Membranes

In this section we show that every RE language can be generated by a RP system (with priority and without dissolving action) with exactly two rules in each region and of depth two.

In the following proof 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 each sequence in M is called matrix). 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.

Definition 1. *A RP system is in double_2_ normal_form if it is of depth 2 and in each membrane we have exactly 2 rewriting rules.*

Theorem 1. *Every RE language can be generated by a RP system (with electrical charges, with priority and without dissolving action) in double_2_normal_form.*

Proof. Consider a matrix grammar with appearance checking $G = (N, T, S, P, F)$ in the normal form previously described. We assume the matrices labeled in a one-to-one manner. With $m_1, \ldots, m_{k_1}$ we label the matrices of type 2, with $m_{k_1+1}, \ldots, m_{k_2}$ we label the matrices of type 3 and with $m_{k_2+1}, \ldots, m_k$ we label the matrices of type 4. Moreover, we label the symbols in N with $B_1, \ldots, B_h$.

We show how to construct a P system of depth 2 with exactly 2 rewriting rules in each membrane that generates the same language of G:

$$\Pi = (V, \mu, M_0, M_1, \ldots, M_k, M_{k+1}, \ldots, M_{k+h+1}, \\ (R_0, \rho_0), (R_1, \rho_1), \ldots, (R_{k+h+1}, \rho_{k+h+1}), 0)$$

where

- $V = N_1 \cup N_2 \cup \{Z, Z', Z''\} \cup \{C_i | 1 \leq i \leq h\} \cup T$,
- $\mu = [_0 [_1]^+_1 \ldots [_k]^+_k [_{k+1}]^-_{k+1} \ldots [_{k+h+1}]^-_{k+h+1}]_0$
- $M_0 = \{Z' ZXA | S \to XA$ is the rule of a matrix of type 1$\}$
- $M_1, ..., M_{k+h+1}$ are empty
- $R_0 = \{r_{0,1} : Z \to \lambda(-)\} \cup \{r_{0,2} : Z' \to Z'(+)\}$
- $R_\alpha = \{r_{\alpha,1} : X \to ZY\} \cup \{r_{\alpha,2} : A \to x(out)\}$, $1 \leq \alpha \leq k_1$, ($m_\alpha : (X \to Y, A \to x)$ is a type 2 matrix)
- $R_\beta = \{r_{\beta,1} : A \to A\} \cup \{r_{\beta,2} : X \to ZY(out)\}$, $k_1 + 1 \leq \beta \leq k_2$, ($m_\beta : (X \to Y, A \to \dagger)$ is a type 3 matrix)
- $R_\gamma = \{r_{\gamma,1} : X \to C_1\} \cup \{r_{\gamma,2} : A \to x(out)\}$, $k_2 + 1 \leq \gamma \leq k$, ($m_\gamma : (X \to \lambda, A \to x)$ is a type 4 matrix)
- $R_\psi = \{r_{\psi,1} : B_{\psi-k} \to B_{\psi-k}\} \cup \{r_{\psi,2} : C_{\psi-k} \to C_{\psi-k+1}(out)\}, k + 1 \leq \psi \leq k + h - 1$,
- $R_{k+h} = \{r_{k+h,1} : B_h \to B_h\} \cup \{r_{k+h,2} : C_h \to Z''(out)\}$,
- $R_{k+h+1} = \{r_{k+h+1,1} : Z' \to \lambda\} \cup \{r_{k+h+1,2} : Z'' \to \lambda(out)\}$
- $\rho_i : r_{i,1} > r_{i,2}$, for every $0 \leq i \leq h + k + 1$

In other words, we place into the skin membrane several membranes with positive charge, one membrane for each matrix in G; each membrane simulates the productions of a matrix in G. Moreover, we place into the skin membrane several membranes with negative charge, one membrane for every nonterminal symbol in G, used to verify that the generated strings do not contain non terminal symbols. Finally we put one further membrane, with negative charge, to stop the computation.

Consider a string of the form $Z^{'}ZHw$ in membrane 0 with $w \in (N_2 \cup T)^*$ and $H \in N_1$ (initially we have $Z^{'}ZXA$). We have to apply the production $Z \to \lambda(-)$ and we get the string $Z^{'}Hw$. This string is sent to a membrane with positive charge, in which we simulate the productions of a type 2, 3 or 4 matrix.

Membrane simulating a type 2 matrix If the string is sent to a membrane corresponding to a type 2 matrix, we have to apply a rule of the form $X \to ZY$ (which simulates the first production of the type 2 matrix) and a rule of the form $A \to x(out)$ (which simulates the second production of the type 2 matrix). The first production of a matrix of type 2 in the binary normal form cannot be of the form $X \to X$, as one can see from the description of the normal form in [2]. Thus, if the symbol X is in the string (i.e. $H = X$), we have to apply this rule and we can do it only one time (the string cannot evolve forever in this membrane due to a rule $X \to X$). Otherwise, we cannot apply this rule and the symbol Z is not reinserted in the string. Then, we have to apply the rule $A \to x(out)$. If the symbol A is not in w, the string cannot further evolve and it will not reach the output membrane; otherwise the obtained string is sent back in the skin membrane. As said before, if the production $X \to ZY$ has not been applied, we have now a string of the form $Z^{'}Hw^{'}$, i.e. a string without the symbol Z. Thus, in the skin membrane we have to apply the rule $Z^{'} \to Z^{'}(+)$, which sends the string into a membrane with negative charge. It easy to see that there is no such a membrane which sends back the string in the skin membrane (the string does not contain the symbol $Z^{''}$ nor a symbol C_i). The computation can correctly proceed only if the membrane correctly simulates the corresponding type 2 matrix. In fact, if we apply the production $X \to ZY$ and if the symbol A is in w, we can apply the rule $A \to x(out)$, that sends back in membrane 0 the string $Z^{'}ZYw^{'}$, that is ready to simulate another matrix.

Membrane simulating a type 3 matrix If the string $Z^{'}Hw$ is sent to a membrane of type 3, we have to apply the rules $A \to A$ and $X \to ZY$. We have the following possibilities: if the string contains the symbol A, we have to apply forever the production $A \to A$ (due to the priority), thus the computation will never stop and no string will be produced. If the symbol A is not in the string, we can apply the production $X \to ZY(out)$. If $H \neq X$ the string cannot further evolve and it will remain in this membrane forever, otherwise we correctly simulate a type 3 matrix and the string is sent back in membrane 0, where we can start the simulation of another matrix.

Membrane simulating a type 4 matrix If the string $Z^{'}Hw$ is sent to a membrane of type 4, we have to apply the rules $X \to C_1$ and $A \to x(out)$. If the string does not contain the symbol A, it will remain in the membrane forever. If the string contains the symbol A but it does not contain the symbol X, we can apply the rule $A \to x(out)$. The obtained string does not contain the symbol Z nor the symbol C_1. As we have seen before for the type 2 matrix, this string will reach a membrane with negative charge but it will never exit from that, thus no string will be generated in this way. The matrix will be correctly simulated only if $H = X$ and the string contains the symbol A. In this case, we first apply

the rule $X \to C_1$ and then the rule $A \to x(out)$. In the skin membrane we get a string of the form $Z^{'}C_1w$.

Thus, we are able to simulate the productions of every type of matrix in the correct order. When the string reaches a membrane that corresponds to a type 4 matrix, the phase of simulating the production of the matrices has to be ended, and we have to control that the obtained string does not contain non terminal symbols. This is done with the negative charged membranes. Consider a string of the form $Z^{'}C_1w$ in membrane 0. Obviously, the production $Z \to \lambda$ cannot be applied, because the string does not contain the symbol Z. Thus, we have to apply the rule $Z^{'} \to Z^{'}(+)$. The obtained string $Z^{'}C_1w$ is sent to a membrane with negative charge. There is only one membrane with a production that involved C_1: the membrane used to control the presence of the non terminal B_1, that is the membrane $k+1$; it contains the productions $B_1 \to B_1$ and $C_1 \to C_2(out)$. If the string reaches a different membrane, it cannot further evolve and no string reaches the output membrane. Otherwise, we can test the presence of the non terminal B_1 in the string: if B_1 is in the string, we have to apply forever the production $B_1 \to B_1$, otherwise we can apply the production $C_1 \to C_2(out)$ and we send back in membrane 0 the string $Z^{'}C_2w$. Here we can apply again the rule $Z^{'} \to Z^{'}(+)$ to send the string in a membrane with negative charge.

Now, the "correct" one is the membrane that tests the presence of the non terminal symbol B_2. If the string reaches another membrane, it will be blocked. If it reaches the membrane $k+2$ and the string contains the symbol B_2, the computation will never halt, otherwise we enter membrane 0 with the string $Z^{'}C_3w$. The computation proceeds in this way until we test all non terminal symbols. The sequence $C_1, C_2, \ldots, C_h$ permits us to be sure that we test the presence of all non terminal symbols.

In the membrane $k+h$ (which checks the presence of the $h-th$ non terminal symbol) we have the production $C_h \to Z^{''}(out)$ ($Z^{''}$ tell us that we have checked the presence of all non terminal symbols). The string is sent back in membrane 0 where we have to apply again the rule $Z^{'} \to Z^{'}(+)$. The string $Z^{'}Z^{''}w$ is sent to a membrane with negative charge. If it reaches the membrane $k+h+1$ we apply the rules $Z^{'} \to \lambda$ and $Z^{''} \to \lambda(out)$ that sends back in membrane 0 (the output one) the terminal string w; otherwise the string will be blocked in one of the other membrane with negative charge.

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

As previously said, the system in the proof takes advantage of 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 allows one to control the communication of the strings with only two general rules (in the skin membrane), because we do not have to specify the label of the membrane where the strings have to go (as in the models presented in [5]). With one rule we simulate a matrix and with the other we stop the simulation and we 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 in the same membrane, as we have shown in the proof.

Note that the proof presented here does not show how to build a system in double_2_normal_form starting from a generic RP systems with priority (in a direct way). Of course, given a RP system, we can build an equivalent type 0 grammar, from this we can build the equivalent matrix grammar with appearance checking and finally we can obtain an equivalent RP system in the normal form. It would be interesting to show how to obtain a system in the normal form in a direct way.

4 Rewriting P Systems without Priority

We consider now RP systems without priority and with *External Output*. This last feature were introduced in [9] and the differences with respect to the model previously described are the following:

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

We show that such a system can be reduced to a system with three rules in each membrane.

In the following we denote with $RPE^{\pm}(nPri, n\delta)$ the family of languages generated by Rewriting P systems with external output, which use electrical charges and which do not use priority relations nor dissolving membrane action.

Definition 2. *A P system is in 3_ normal_form if in each membrane we have exactly 3 rewriting rules.*

Theorem 2. . *Every language $L \in RPE^{\pm}(nPri, n\delta)$ can be generated by a RP system of the same type in 3_normal_form.*

Proof. Consider a RP system with external output

$$\Pi = (V, \mu, M_1, \ldots, M_k, R_1, \ldots, R_k)$$

We denote with m_i the $i-th$ membrane, and we assume that the skin membrane is m_1.

We show how to construct a RP system

$$\Pi^{'} = (V^{'}, \mu^{'}, M_1^{'}, \ldots, M_{k'}^{'}, R_1^{'}, \ldots, R_{k'}^{'})$$

($V' = V \cup \{Z, Z_o, Z'_o, Z_i, Z_+, Z_-\}$) in 3_normal_form, which generates the same language of Π.

To explain how we build Π', consider a membrane m_i of Π, with $i \neq 1$. Such a membrane contains a set of string M_i and a set of rules R_i. Moreover there are a number of cells placed immediately inside this one, some with positive charge and others with negative charge.

First of all, we replace every string w in m_i with a string Zw, where Z is a symbol not in V. We will use this symbol and the symbols $Z_o, Z_{o'}, Z_i, Z_+, Z_-$ to control the travel of the strings between the membranes. These symbols will be deleted before the string leaves the skin membrane, thus their position into the strings is of no interest (we need this later in the proof).

Then we place a membrane around the cells placed immediately inside m_i, in such a way that the rules in m_i remain outside this new membrane while the membranes in m_i will be inside this new membrane. We label this membrane with a_i (to indicate the membrane Around the membrane in i) and we give it a positive charge. In the new region just created, we place the rules:

- $Z_+ \to Z(+)$, $Z_- \to Z(-)$, $Z'_o \to Z(out)$

Then, we consider the rules in R_i. For every rule in R_i, we add a new membrane in the region defined by R_i and by the new membrane a_i, and we give negative charge to these membranes. If R_i contains q rules, we add q membranes. Each membrane will be used to simulate a rule. We denote these membranes with $mr_{i,j}$, where $1 \leq j \leq q$ (to indicate the Membrane added in i to simulate the Rule j). We have to simulate the rule and the travel of the obtained string too. For this reason, we get four different form of membrane (here, out, positive charge, negative charge).

For every rule of the form $A \to x(here)$ we add a membrane with the rules:

- $A \to Zx(out)$, $A \to ZA(out)$, $A \to A$

For every rule of the form $A \to x(out)$ we add a membrane with the rules :

- $A \to Z_o x(out)$, $A \to ZA(out)$, $A \to A$

For every rule of the form $A \to x(+)$ we add a membrane with the rules :

- $A \to Z_i Z_+ x(out)$, $A \to ZA(out)$, $A \to A$

For every rule of the form $A \to x(-)$ we add a membrane with the rules :

- $A \to Z_i Z_- x(out)$, $A \to ZA(out)$, $A \to A$

No string is initially present in these new membranes.

Note that in each membrane there is only one rule (the first one) that effectively simulates the corresponding rule in m_i. The two other rules are dummy rules introduced to get a system with exactly three rules in each membrane. As we will see later in the proof, they have no influence on the strings.

Once we have created these new membranes, we delete all the rule in m_i and we place in this membrane the rules:

– $Z \to \lambda(+)$, $Z_i \to \lambda(-)$, $Z_o \to Z_o^{'}(out)$

The membrane obtained in this manner is denoted by $m_i^{'}$ (to indicate that the membrane has a corresponding membrane m_i in Π, and it is not a new one).

Thus, all membrane except the skin membrane are changed as described above. We modify the skin membrane in the same way with only one difference: the rule $Z_o \to Z_o^{'}(out)$ will not be added in $m_1^{'}$. Instead, we add the rule $Z_o \to \lambda(out)$.

To see how the computation proceeds, consider a string w in a membrane m_i. Using a rule in R_i we can replace a symbol in w to obtain a string w'. Then, w' can remain in the same membrane and it can be involved in a new rewriting rule or it can be sent to the membrane immediately outside or w' can be sent to a membrane immediately inside m_i (with positive or negative charge).

In the corresponding membrane of $\Pi^{'}$ (i.e. in $m_i^{'}$) we have the string Zw; on this string we can apply only the rule $Z \to \lambda(+)$; the obtained string w will be sent to a membrane $mr_{i,j}$, where we simulate one rewriting rule and we define where the obtained string have to be sent:

HERE If w reaches a membrane corresponding to a rule $A \to x(here)$ and w does not contain the symbol A, the string will remain in that membrane forever. Otherwise, we have the following possibilities:

– We can apply the rule $A \to Zx(out)$ to simulate the corresponding rule in m_i. The obtained string is of the form $w_1 Z w_2$, where $w_1 w_2 = w'$, and it will be sent outside to the membrane $m_i^{'}$, where we can apply again the rule $Z \to \lambda(+)$.
– We can apply the rule $A \to ZA(out)$. The obtained string is of the form $w_3 Z w_4$, where $w_3 w_4 = w$; it will be sent outside to the membrane $m_i^{'}$. The only difference between this string and the string Zw is the position of Z but, as said before, it is not important for the computation. Thus, this rule has no effect on the string.
– We can apply the rule $A \to A$. The string is not modified; it can be used many times with this rule, until we apply one of the other two rules, to send the string back in membrane $m_i^{'}$.

OUT If w reaches a membrane corresponding to a rule $A \to x(out)$ and w does not contain the symbol A, the string will remain in that membrane forever. Otherwise, we have the following possibilities:

– We can apply the rule $A \to Z_o x(out)$ to simulate the corresponding rule in m_i. The obtained string is of the form $w_1 Z_o w_2$, where $w_1 w_2 = w'$, and it will be sent back to membrane $m_i^{'}$. In membrane $m_i^{'}$ we have two possibilities:
  - If $i \neq 1$ (i.e. $m_i^{'}$ is not the skin membrane), we have to apply the rule $Z_o \to Z_o^{'}(out)$. We send the string $w_1 Z_o^{'} w_2$ to the membrane immediately outside $m_i^{'}$, which is a membrane with a label a_v (v is the label of the membrane immediately outside the membrane i in Π). The membrane a_v is one of the membrane added in $\Pi^{'}$, originally not in Π. In membrane

a_v, we can only apply the rule $Z_o^{'} \rightarrow Z(out)$, which sends the string $w_1 Z w_2$ to the membrane outside, which is the membrane $m_v^{'}$. Thus, we correctly simulate the application of the rule and the obtained string is sent to the correct membrane, where we can start the simulation of another rule.

- If $i = 1$ (i.e. $m_i^{'}$ is the skin membrane), we have to apply the rule $Z_o \rightarrow \lambda(out)$. The symbol Z_o is deleted and the obtained string is sent outside the system; thus, if the string is a terminal one, it will be in the language $L(\Pi^{'})$.

– We can apply the rule $A \rightarrow ZA(out)$ or the rule $A \rightarrow A$. The consideration for the HERE case is still valid.

POSITIVE CHARGE If w reaches a membrane corresponding to a rule $A \rightarrow x(+)$ and w does not contain the symbol A, the string will remain in that membrane forever. Otherwise, we have the following possibilities:

– We can apply the rule $A \rightarrow Z_i Z_+ x(out)$ to simulate the corresponding rule in m_i. The obtained string is of the form $w_1 Z_i Z_+ w_2$, where $w_1 w_2 = w'$, and it will be sent outside to membrane $m_i^{'}$. Here we delete the symbol Z_i with the rule $Z_i \rightarrow \lambda(-)$. The string will be sent to the membrane with positive charge in $m_i^{'}$, i.e. the membrane a_i. In a_i we can apply only the rule $Z_+ \rightarrow Z(+)$. The obtained string $w_1 Z w_2$ will be sent to a membrane with negative charge, and the simulation of the rule $A \rightarrow x(+)$ is correctly done.
– We can apply the rule $A \rightarrow ZA(out)$ or the rule $A \rightarrow A$. The consideration for the HERE case is still valid.

NEGATIVE CHARGE This case is similar to the previous one.

Consider now the case that no rule in m_i (membrane of Π) can be applied to w. The string will remain blocked in m_i forever, thus no string will be generated from it. In the corresponding membrane $m_i^{'}$ (membrane of $\Pi^{'}$) we have the string Zw. We can apply the rule $Z \rightarrow \lambda(+)$ and the string w will be sent to a membrane $mr_{i,j}$. Because no rule in m_i can be applied to w, there is no membrane $mr_{i,j}$ with a rule that can be applied to it (the left symbol of the productions in $mr_{i,j}$ is the left symbol of a production in m_i), so the string w will remain blocked forever in the membrane.

Another possibility is that the computation on w in Π never halts, thus no string will be generated from w. In $\Pi^{'}$ we have two possibilities: the computation can continue forever by simulating the same rules or it can end, when the string w reaches a membrane $mr_{a,b}$ with no rules applicable on w. In both cases no string will be generated from Zw.

Thus, the strings that leave the skin membrane in $\Pi^{'}$ are the same of those in Π, that is $L(\Pi) = L(\Pi^{'})$. □

5 Conclusions

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

We have made 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 that are easy to analyze, in order to understand their generative power and their equivalence with other generative mechanisms. For instance, we have shown that every RP system of depth two and with two rules in each membrane can generate every RE language, by showing that such a system is equivalent to a Matrix Grammar System with Appearance Checking; moreover, we have shown that every RP system without priority is equivalent to a RP system with exactly three rules in each membrane. It is not known if RP systems without priority are able to generate RE language or not; we hope that the normal forms given here can be useful to give an answer to this question.

References

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

[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 (1994), 719-728.

[4] Gh Paun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (Febr. 1999), 139-152.

[5] Gh. Paun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108-143, and *Turku Center for Computer Science-TUCS Report No 208*, 1998 (www.tucs.fi).

[6] Gh. Paun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182, and *Auckland University, CDMTCS Report No 098*, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[7] Gh. Paun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266, and *Turku Center for Computer Science-TUCS Report No 218*, 1998 (www.tucs.fi).

[8] Gh. Paun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397–410, and *University of Western Ontario Report TR 539*, 1999 (www.csd.uwo.ca/faculty/syu/TR539.html).

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

[10] I. Petre, A normal form for P systems, *Bulletin of EATCS*, 67 (Febr. 1999), 165-172.

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