

Using Membrane Features in P Systems*

C. Zandron, C. Ferretti, G. Mauri

DISCO - Università di Milano-Bicocca - Italy

Abstract

In the basic variant of P systems, membranes are used as separators and as channels of communication. Other variants, introduced to obtain more “realistic” models, consider membranes with different features: membranes of variable thickness, electrically charged membranes and active membranes (membranes can be divided in two or more membranes). These features are not only useful to obtain “realistic” models: we show how we can use them to get simpler and faster models.

1 Introduction

The P systems were recently introduced in [6] 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.

In this basic variant, the membranes are used only as separators of objects and as channels of communication. In [7] and [8] new features are introduced:

- **Membranes of variable thickness:** membranes can be made thicker or thinner (also dissolved as said before). Initially, all membranes have thickness 1. If a membrane has thickness 2, then no object can pass through it.

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

- **Membranes with electrical charges:** electrical charges are associated to membranes and to objects: they can be marked with “positive” (+), “negative” (−) or “neutral” (0). The charge define the communication of the objects: an object marked with + (respectively −) will enter a membrane marked with − (respectively +), nondeterministically chosen from the set of membranes adjacent to the region where the object is produced. The neutral objects are not introduced in an inner membrane. Thus, the evolution rules do not specify the label of the membrane where the object will be sent, but they only associate a charge to every object involved in the application of the rule itself.
- **Active membranes:** the membranes can not only be dissolved, but they can multiply by division. We consider here division rules for elementary membranes only, i.e. membranes not containing other membranes (in [8] one considers division for non-elementary membranes too). Starting from a membrane we get two (or more) membranes, each with the same objects and rules of the original membrane.

Other variants are considered in [1], [6], [9] and [10].

In this paper we show how to use the previous features to get simpler and faster models of P systems. In particular we show how to:

- Use variable thickness to simulate priority in (Rewriting) P Systems.
- Use electrical charges to simplify membrane structures in (Rewriting) P systems.
- Use membrane division for elementary membranes to execute fast computations.

2 P systems and Rewriting P systems

We give a very short definition of P system; details and examples can be found in [5], [6] and [7]. We refer to [13] 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 the membrane immediately inside it. If in the regions we place multi sets 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.

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

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

where:

- V is an alphabet
- μ is a membrane structure consisting of n membranes (labeled with $1, \dots, n$)
- $M_i, 1 \leq i \leq n$ are multi sets over V associated with the regions $1, 2, \dots, n$ of μ ;
- $R_i, 1 \leq i \leq n$ are finite sets of evolution rules associated with the regions $1, 2, \dots, n$ of μ ; ρ_i is a partial order relation over $R_i, 1 \leq i \leq n$, specifying a priority relation among rules of R_i . The rules are of the form $u \rightarrow v$ where u is a symbol of V and $v = v'$ or $v = v'\delta$. v' is a string over $(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq n\})$ and δ is a special symbol not in V . When we apply a rule containing the symbol δ , the membrane where the rule is applied is dissolved. The rules of that membrane are deleted and the objects remain free in the membrane placed immediately outside. The skin membrane is never dissolved.
- i_0 is the output membrane. If we don't specify the output membrane, we consider as output the objects sent out from the skin membrane in the order of their expelling from the system (this variant was introduced in [9]).

We consider here not only P systems but *Rewriting P Systems* (RP Systems) too. 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 strings. Consequently, the evolution rules are given as rewriting rules. We only use context-free rewriting rules.

We describe now three variants of P systems, which consider membranes with different features:

- **Membranes of variable thickness:** the rules are of the form $u \rightarrow v(tar)$ where u is a symbol of V and $v = v'$ or $v = v'\delta$ or $v = v'\tau$. v' is a string over V , while 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.

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 a membrane has thickness 2, then no object can pass through it. If both the symbols δ and τ are introduced in the same region, the corresponding membrane preserves its thickness.

The communication of objects has priority on 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.

- **Membranes with electrical charges:** μ is a membrane structure consisting of n membranes; each membrane is marked with one of the symbols $+$, $-$, 0 .

The rules are of the form

$$(u \rightarrow v(p))$$

where u 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 $p \in \{here, out, +, -\}$.

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 a membrane marked with $-$ (respectively $+$) and adjacent to the region where the rule is applied.

- **Active (and electrically charged) membranes:** A P system with active membranes is a construct $\Pi = (V, T, H, \mu, w_1, \dots, w_m, R)$ where:

- $m \geq 1$
- V is an alphabet
- $T \subseteq V$ is the terminal alphabet
- H is a finite set of labels for membranes
- μ is a membrane structure consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H ; all membranes in μ are supposed to be neutral
- w_1, \dots, w_m are strings over V , describing the multisets of objects placed in the m regions of μ
- R is a finite set of developmental rules, of the following forms:
 1. Type (a): $[_h a \rightarrow v]_h^\alpha$, for $h \in H, a \in V, v \in V^*, \alpha \in \{+, -, 0\}$ (object evolution rules),
 2. Type (b): $a[_h]_h^{\alpha_1} \rightarrow [_h b]_h^{\alpha_2}$, where $a, b \in V, h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}$ (an object is introduced in membrane h),
 3. Type (c): $[_h a]_h^{\alpha_1} \rightarrow [_h]_h^{\alpha_2} b$, for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$ (an object is sent out from membrane h),
 4. Type (d): $[_h a]_h^\alpha \rightarrow b$, for $h \in H, \alpha \in \{+, -, 0\}, a, b \in V$ (membrane h is dissolved),
 5. Type (e): $[_h a]_h^{\alpha_1} \rightarrow [_h b]_h^{\alpha_2} [_h c]_h^{\alpha_3}$, for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in V$ (division rules for elementary membranes)
 6. Type (f): $[_{h_0} [_{h_1}]_{h_1}^{\alpha_1} \dots [_{h_k}]_{h_k}^{\alpha_1} [_{h_{k+1}}]_{h_{k+1}}^{\alpha_2} \dots [_{h_n}]_{h_n}^{\alpha_2}]_{h_0}^{\alpha_0} \rightarrow$
 $[_{h_0} [_{h_1}]_{h_1}^{\alpha_3} \dots [_{h_k}]_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5} [_{h_0} [_{h_{k+1}}]_{h_{k+1}}^{\alpha_4} \dots [_{h_n}]_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$,
for $k \geq 1, n > k, h_i \in H, 0 \leq i \leq n$, and $\alpha_0, \dots, \alpha_6 \in \{+, -, 0\}$ with $\{\alpha_1, \alpha_2\} = \{+, -\}$ (division rules for non-elementary membranes)

The rules are applied following the principles in [8]. When a membrane is divided by a rule of type (e) or (f), then the content of this membrane is reproduced unchanged in the new copies we get.

In the next chapters we will need the notion of matrix grammar, too. 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 $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$, $n \geq 1$, 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 $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ 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 $A_i \rightarrow x_i$ 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 $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. 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. $(S \rightarrow XA)$, with $X \in N_1, A \in N_2$,
2. $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \rightarrow Y, A \rightarrow \dagger)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \rightarrow \lambda, A \rightarrow x)$, 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 $A \rightarrow \dagger$ 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 [6], 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 [13]. Moreover, in [3] it is shown that the one-letter languages in MAT are regular.

3 Using variable thickness to simulate priority

It is known (see, for ex., [6]) that Rewriting P systems which made use of priority on the evolution rules are able to generate every Recursively Enumerable language. Nevertheless, the priority relation among evolution rules is a feature of a formal language inspiration, which does not seem a "realistic" one.

In the next proof, we prove that if we use the feature that modifies the thickness of a membrane, we are able to generate every RE language with RP systems that do not make use of priority. In fact, by modifying the thickness of the membrane we are able to simulate, in the correct order, the productions of a generic matrix grammar with appearance checking.

With $RP(nPri, \delta, \tau)$, we denote the family of languages generated by Rewriting P systems without priority on the evolution rules and which made use of variable thickness (i.e. both δ and τ operations).

Theorem 1 $RP(nPri, \delta, \tau) = RE$

Proof 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 binary normal form. We assume the matrices of the types 2, 3 and 4 labeled in a one-to-one manner; we label with $m_1, \dots, m_{k'}$ the matrices of type 2, with $m_{k'+1}, \dots, m_{k''}$ the matrices of type 3 and with $m_{k''+1}, \dots, m_k$ 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 :

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

where

- $V = N_1 \cup N_2 \cup \{E, \dagger, F, F', F_2, F_3, F_4\} \cup T \cup \{X', X_2, X_3 | X \in N_1\}$
- $\mu = [0[k+1[1]1[2]2] \dots [k]k]_{k+1}0$
- $M_{k+1} = \{XAE | S \rightarrow XA \text{ rule of the matrix of type 1}\} \cup \{F\}$
- $M_0 = \emptyset$
- $M_h = \dagger, \text{ for } 1 \leq h \leq k$
- $R_i (1 \leq i \leq k') = \{A \rightarrow x\delta(out) | m_i : (X \rightarrow Y, A \rightarrow x) \text{ type 2 matrix, } x \in (N_2 \cup T)^*\} \cup \{F' \rightarrow F\tau(out)\}$
- $R_j (k' < j \leq k'') = \{Y' \rightarrow Y_2\tau\} \cup \{F' \rightarrow F_2\tau\} \cup \{Y_2 \rightarrow Y_3\} \cup \{F_2 \rightarrow F_3\tau\} \cup \{A \rightarrow \dagger | m_j : (X \rightarrow Y, A \rightarrow \dagger) \text{ type 3 matrix}\} \cup \{F_3 \rightarrow F_4\delta\} \cup \{Y_3 \rightarrow Y\delta(out)\} \cup \{F_4 \rightarrow F\tau(out)\}$
- $R_h (k'' < h \leq k) = \{A \rightarrow x_2\delta(out) | m_h : (X \rightarrow x_1, A \rightarrow x_2) \text{ type 4 matrix, } x_1, x_2 \in T^*\} \cup \{F' \rightarrow F\tau(out)\}$
- $R_{k+1} = \{X \rightarrow Y(in_w) | m_w : (X \rightarrow Y, A \rightarrow x) \text{ type 2 matrix, } x \in (N_2 \cup T)^*\} \cup \{X \rightarrow Y'(in_w) | m_w : (X \rightarrow Y, A \rightarrow \dagger) \text{ type 3 matrix}\} \cup \{X \rightarrow x_1(in_w) | m_w : (X \rightarrow x_1, A \rightarrow x_2) \text{ type 4 matrix, } x_1, x_2 \in T^*\} \cup \{E \rightarrow \lambda(out)\} \cup \{F \rightarrow \lambda\} \cup \{F \rightarrow F'(in_r), 1 \leq r \leq k\} \cup \{\dagger \rightarrow \dagger\}$
- $R_0 = \{\alpha \rightarrow \alpha | \alpha \in V - T\}$

The most external membrane (M_0) is the output one. This membrane contains a membrane (M_{k+1}) used to control the simulation of the matrices. Inside this membrane there are k membranes, one for every matrix of the matrix grammar we have to simulate.

Consider the strings XwE (initially we have XAE) and F in membrane M_{k+1} with $w \in (N_2 \cup T)^*$. On the string F we can apply the following rules:

- $F \rightarrow \lambda$. This eliminates the string F .
- $F \rightarrow F'(in_r)$, with $1 \leq r \leq k$. These rules send the string F' in a membrane corresponding to a matrix of the matrix grammar system.

On the string XwE we can apply one of the following rules:

- $E \rightarrow \lambda(out)$. This rule eliminates the symbol E from the string and it sends the obtained string in membrane 0, the output one.
- $X \rightarrow Y(in_w)$, where w is a label of a membrane associated with a type 2 matrix. These rules simulate the first production of a type 2 matrix and they send the string to the corresponding membrane.
- $X \rightarrow Y'(in_w)$, where w is a label of a membrane associated with a type 3 matrix. These rules simulate the first production of a type 3 matrix and they send the string to the corresponding membrane.
- $X \rightarrow x_1(in_w)$, where w is a label of a membrane associated with a type 4 matrix. These rules simulate the first production of a type 4 matrix and they send the string to the corresponding membrane.

Consider what happens if we simulate the first production of a matrix on the string XwE and we send the obtained string in the corresponding membrane i ($1 \leq i \leq k$), and, at the same time, a rule $F \rightarrow F'(in_j)$ with $j \neq i$ or a rule $F \rightarrow \lambda$ on F .

The string obtained from XwE is sent ALONE in membrane i .

If the membrane i is a membrane that simulates a type 2 matrix, we send in that membrane a string of the form YwE . If this string contains the corresponding symbol $A \in N_2$ we have to apply the rule $A \rightarrow x\delta(out)$. The membrane is dissolved, thus the string \dagger , present in the membrane, reaches the membrane $k+1$ where we have the rule $\dagger \rightarrow \dagger$: the computation will never stop. If the string YwE doesn't contain a symbol $A \in N_2$, the string cannot further evolve and, as we will see, no string can reach the output membrane.

If the membrane i is used to simulate a type 4 matrix, the situation is similar. The only difference is in the string we send: it is of the form x_1wE , where $x_1 \in T^*$.

Finally, if i corresponds to a type 3 matrix, we send in the membrane a string of the form $Y'wE$. We can either apply a rule $Y' \rightarrow Y_2\delta$, which dissolves the membrane and, consequently, sends the string \dagger in membrane $k+1$, or, if the string contains the corresponding symbol $A \in N_2$, we can apply the rule $A \rightarrow \dagger$ that introduces the trap symbol \dagger into the string. In both cases, the computation will never halt or no string will reach the output membrane.

The only way to correctly simulate a matrix is to apply, at the same time, the rules that send in the same membrane the strings obtained from XwE and F .

To simulate a type 2 matrix we have to apply a rule $X \rightarrow Y(in_i)$ on XwE and a rule $F \rightarrow F'(in_i)$ on F . In this way, we get the strings YwE and F' . Then these strings are sent to membrane i .

In membrane i , we have to apply the rule $F' \rightarrow F\tau$ on F' and the rule $A \rightarrow x\delta(out)$ on YwE ; in this way we send back in membrane $k + 1$ the strings F and Yw_1xw_2E (with $w_1Aw_2 = w$). The thickness of the membrane remains unchanged (one rule uses the symbol δ while the other rule uses the symbol τ). Thus, we have correctly simulated the productions of a type 2 matrix on XwE (and the thickness of membrane i is still 1); we can proceed with the simulation of another matrix.

Note that if the symbol A is not present in XwE , this string can never leave the membrane, thus it cannot reach the output membrane.

To simulate a type 4 matrix, the process is quite similar. The only difference is that we have to apply a rule $X \rightarrow x_1(in_i)$ on XwE , where i is a label of a membrane used to simulate a type 4 matrix and x_1 is a terminal string.

The process of simulating a type 3 matrix is quite different. To simulate the productions of such a matrix we have to apply a rule $X \rightarrow Y'(in_i)$ on XwE and the rule $F \rightarrow F'(in_i)$ on F . We get the strings $Y'wE$ and F' and these strings are sent to membrane i . Here, we have a 4 step process:

Step 1 (we control if the string $Y'wE$ is arrived alone).

On $Y'wE$ we can apply either the rule $Y' \rightarrow Y_2\delta$ or the rule $A \rightarrow \dagger$ (if $Y'wE$ contains the symbol A). At the same time, on F' we have to apply the rule $F' \rightarrow F_2\tau$.

If we apply the rule $A \rightarrow \dagger$ on $Y'wE$, we introduce the trap symbol \dagger into the string; even if the string will reaches the output membrane, the computation will never halt. Otherwise, we get the strings Y_2wE and F_2 . The thickness of the membrane remains unchanged (one rule uses the symbol δ while the other uses the symbol τ).

Step 2 (we increment the thickness of membrane).

On the string of the form Y_2wE we can apply either the rule $A \rightarrow \dagger$ or the rule $Y_2 \rightarrow Y_3$. At the same time, on F_2 we have to apply the rule $F_2 \rightarrow F_3\tau$.

If we apply the rule $A \rightarrow \dagger$ on Y_2wE , the considerations of the previous step are still valid. Otherwise, we get the strings Y_3wE and F_3 . The thickness of the membrane is now 2, due to the symbol τ in the rule $F_2 \rightarrow F_3\tau$.

Step 3 (we execute the appearance checking on the string Y_3wE).

On the string of the form Y_3wE we can apply only the rule $A \rightarrow \dagger$ (if the string contains the symbol A). We cannot apply the rule $Y_3 \rightarrow Y\delta(out)$ because the thickness of the membrane is 2, thus the string cannot pass through the membrane. At the same time, on F_3 we have to apply the rule $F_3 \rightarrow F_4\delta$.

Consequently, if a symbol A is present in Y_3wE , we have to apply the rule $A \rightarrow \dagger$ that introduces the trap symbol. If A is not present, no rule can be applied on this string. The thickness of the membrane return to 1, due to the symbol δ in $F_3 \rightarrow F_4\delta$.

Step 4 (we conclude the simulation of a type 3 matrix).

The thickness of the membrane is now 1, so, on the string of the form Y_3wE , we can apply the rule $Y_3 \rightarrow Y\delta(out)$. At the same time, on F_4 we have to apply the

rule $F_4 \rightarrow F\tau(out)$.

The thickness of the membrane remains unchanged. We send back in membrane $k+1$ the string F and a string of the form YwE in which we have correctly simulated a type 3 matrix. We can start the simulation of another matrix.

We conclude this proof by illustrating the use of the rules $E \rightarrow \lambda(out)$ and $F \rightarrow \lambda$.

If we apply the rule $F \rightarrow \lambda$ on F and, at the same time, a rule that simulates the first production of a matrix on XwE , we delete the string F and we send the other string to a membrane labelled with a number between 1 and k . As we previously shown, if this second string is sent alone to a membrane with a label between 1 and k , the computation will never halt.

If we apply the rule $E \rightarrow \lambda(out)$ on a string of the form XwE , we send a string of the form Xw to the output membrane. The computation will never stop due to the rule $X \rightarrow X$ in that membrane. The same is true if we apply such a rule to a string of the form wE in which w contains non terminal symbols.

Thus, consider a string of the form vE in which v is a terminal string (we can obtain such a string after the simulation of a type 4 matrix). If we apply the rule $E \rightarrow \lambda(out)$ on vE we send the terminal string v in the output membrane; no other rule is applied on this string.

In membrane $k+1$ we can apply on F one of the rules $F \rightarrow F'(in_i)$ or the rule $F \rightarrow \lambda$. If we apply a rule $F \rightarrow F'(in_i)$ we send in membrane i a string F' . On this string we have to apply the rule $F' \rightarrow F\tau(out)$ (if i is a label of a membrane that simulates a type 2 or type 4 matrix) or the rules $F' \rightarrow F_2\tau, F_2 \rightarrow F_3\tau, F_3 \rightarrow F_4\delta, F_4 \rightarrow F\tau(out)$ (if i is a label of a membrane that simulates a type 3 matrix). As one can see, in both cases we send back in membrane $k+1$ the string F and the thickness of the membrane i becomes 2.

On F we can apply now the rule $F \rightarrow \lambda$ or one of the rules $F \rightarrow F'(in_j)$, with $j \neq i$, because the membrane i has thickness 2. If we apply one of the rules of the second type, we get the same string F in membrane $k+1$ and another membrane becomes of thickness 2.

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 do not have applied the rule $F \rightarrow \lambda$, we have to apply it. The string F disappear and the computation stops.

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

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 correctly proceed. Otherwise a membrane will be dissolved and a string \dagger will 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.

The difficult in simulating the production of a matrix grammar with appearance checking lies in the type 3 matrix: we have to apply the productions that introduce

the trap symbol only if there are certain symbols in the string; otherwise these productions do not have to be applied. This can be easily controlled using priorities, but it seems difficult to get the same result without priorities. Nevertheless, by modifying the thickness of the membranes, we are able to simulate priority, as illustrated in the proof.

4 Using electrical charges to simplify membrane structure

Another feature in membrane systems which does not seem of a natural inspiration is relative to the communication of objects through membranes: the rules have to specify the label of the membrane where the objects have to be sent after the application of the rule itself. A less restrictive feature consider, as previously described, electrical charges associated with both the objects and the membranes.

We show now that this feature, introduced with the goal of obtaining more realistic systems, can be useful under different aspects. For example, it allows us to simplify the structure of the membrane systems: in fact, the communication of the strings can be controlled with few general rules, because we do not have to specify the precise label of the membrane where the strings have to go (as in the models presented in [6]), but only the subset of membranes we are interested in: positive or negative ones (of course, we have to control that the objects do not reach “wrong” membranes). We can, in this way, define P systems with a limited number of rules per membrane and with a membrane structure of limited depth.

We illustrate this in the following, using Rewriting P system with priority. With $RP^\pm(Pri, n\delta)$ we denote the family of languages generated by Rewriting P systems with electrical charges which use priority on evolution rules and which do not use the operation which allows to dissolve the membranes.

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 1 Every RE language can be generated by a P system $RP^\pm(Pri, n\delta)$ 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, \dots, m_{k_1} we label the matrices of type 2, with $m_{k_1+1}, \dots, m_{k_2}$ we label the matrices of type 3 and with m_{k_2+1}, \dots, m_k we label the matrices of type 4. Moreover, we label the symbols in N with B_1, \dots, B_h .

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

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

where

- $V = N_1 \cup N_2 \cup \{Z, Z', Z''\} \cup \{C_i | 1 \leq i \leq h\} \cup T,$
- $\mu = [0]_1^+ \dots [k]_k^+ [k+1]_{k+1}^- \dots [k+h+1]_{k+h+1}^- 0$

- $M_0 = \{Z'ZXA|S \rightarrow XA \text{ is the rule of a matrix of type 1}\}$
- M_1, \dots, M_{k+h+1} are empty
- $R_0 = \{r_{0,1} : Z \rightarrow \lambda(-)\} \cup \{r_{0,2} : Z' \rightarrow Z'(+)\}$
- $R_\alpha = \{r_{\alpha,1} : X \rightarrow ZY\} \cup \{r_{\alpha,2} : A \rightarrow x(out)\}$, $1 \leq \alpha \leq k_1$, ($m_\alpha(X \rightarrow Y, A \rightarrow x)$ type 2 matrices)
- $R_\beta = \{r_{\beta,1} : A \rightarrow A\} \cup \{r_{\beta,2} : X \rightarrow ZY(out)\}$, $k_1 + 1 \leq \beta \leq k_2$, ($m_\beta(X \rightarrow Y, A \rightarrow \dagger)$ type 3 matrices)
- $R_\gamma = \{r_{\gamma,1} : X \rightarrow C_1\} \cup \{r_{\gamma,2} : A \rightarrow x(out)\}$, $k_2 + 1 \leq \gamma \leq k$, ($m_\gamma(X \rightarrow \lambda, A \rightarrow x)$ type 4 matrices)
- $R_\psi = \{r_{\psi,1} : B_{\psi-k} \rightarrow B_{\psi-k}\} \cup \{r_{\psi,2} : C_{\psi-k} \rightarrow C_{\psi-k+1}(out)\}$, $k + 1 \leq \psi \leq k + h - 1$,
- $R_{k+h} = \{r_{k+h,1} : B_h \rightarrow B_h\} \cup \{r_{k+h,2} : C_h \rightarrow Z''(out)\}$,
- $R_{k+h+1} = \{r_{k+h+1,1} : Z' \rightarrow \lambda\} \cup \{r_{k+h+1,2} : Z'' \rightarrow \lambda(out)\}$
- $\rho_i : r_{i,1} > r_{i,2}$, for every $0 \leq i \leq h + k + 1$

In other words, we place into the skin membrane several membranes with positive charge, one 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 for every nonterminal symbol in G plus one used to stop the computation; these membranes are used to be sure that the generated strings do not contains non terminal symbols.

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 $Z \rightarrow \lambda(-)$ and we get the string $Z'Hw$. This string is sent to a membrane of 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 $X \rightarrow ZY$ (which simulates the first production of the type 2 matrix) and a rule of the form $A \rightarrow x(out)$ (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 $X \rightarrow X$, 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 $X \rightarrow X$). Otherwise, we cannot apply this rule and the symbol Z is not reinserted in the string. Then, we have to apply the rule $A \rightarrow x(out)$. 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 $X \rightarrow ZY$ has not been applied, we have now a string of the form $Z'Hw'$, i.e. a string without the symbol Z in it. Thus, in the skin membrane we have to apply the rule $Z' \rightarrow Z'(+)$, which sends the string in a membrane with negative

charge. It is easy to see that there is no such a membrane that sends back the string in the skin membrane (the string does not contain the symbol Z'' nor a symbol C_i). The computation can proceed correctly only if the membrane correctly simulates the corresponding type 2 matrix. In fact, if we apply the production $X \rightarrow ZY$ and if the symbol A is in w , we can apply the rule $A \rightarrow x(out)$, 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 $A \rightarrow A$ and $X \rightarrow ZY$. We have the following possibilities: if the string contains the symbol A , we have to apply forever the production $A \rightarrow A$ (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 $X \rightarrow ZY(out)$. If $H \neq X$ the string cannot further evolve and it will remain in this membrane, 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 $X \rightarrow C_1$ and $A \rightarrow x(out)$. 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 doesn't contain the symbol X , we can apply the rule $A \rightarrow x(out)$. The obtained string doesn't 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 $X \rightarrow C_1$ and then the rule $A \rightarrow x(out)$. 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 $Z \rightarrow \lambda$ cannot be applied, because the string does not contain the symbol Z . Thus, we have to apply the rule $Z' \rightarrow Z'(+)$. 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 $B_1 \rightarrow B_1$ and $C_1 \rightarrow C_2(out)$. 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 $B_1 \rightarrow B_1$, otherwise we can apply the production $C_1 \rightarrow C_2(out)$ and we send back in membrane 0 the string $Z'C_2w$. Here we can apply again the rule $Z' \rightarrow Z'(+)$ 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 get in membrane 0 the string $Z' C_3 w$. The computation proceeds in this way until we test all non terminal symbols. The sequence C_1, C_2, \dots, C_h permits to be sure that we test the presence of all non terminal symbols. In the membrane that controls the presence of the h-th non terminal symbol (the last one) we have the production $C_h \rightarrow Z''(out)$ (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 $Z' \rightarrow Z'(+)$. The string $Z' Z'' w$ is sent to a membrane with negative charge. If it reaches membrane $k + h + 1$ we apply the rules $Z' \rightarrow \lambda$ and $Z'' \rightarrow \lambda(out)$ 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 $L(G) = L(\Pi)$. \diamond

Informally, the communication of the strings (from the skin membrane to the other membranes) is accomplished using two general rules, because we do not have to specify the label of the membrane where we send the string: with one rule we start the simulation of a matrix, while with the other rule we stop the simulation and we start the phase in which we control if the string is a terminal one. We can control if a string reaches a “wrong” membrane using the rules in the same membrane, as we’ve 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.

5 Using membrane division to solve NP complete problems in polynomial time with P systems

Could we use P systems to solve complex problems in an efficient way? In [8] one considers a variant of P systems in which a membrane can be divided in two membranes; each new membrane contains the same objects and rules of the starting membrane. In [4] one considers the division of a membrane in an arbitrary finite number of membranes. In [8] and [4] it is shown how to solve in linear time (with respect to the input length) two well known NP Complete problems, Satisfiability and Hamiltonian Path, using P systems with active membranes. The systems in these papers use division for elementary membranes and for non elementary membranes (i.e. membranes with one or more membranes inside).

We show now that division for elementary membranes suffice to build P systems able to solve complex problems in an efficient way. In particular, we show how to build a P system, with division for elementary membranes only (called P Systems with Elementary Active Membranes), able to solve the SAT problem in linear time.

Theorem 3: The SAT problem can be solved in linear time (with respect to the number of variables and the number of clauses) by a P system with elementary active membranes.

Proof: Consider a boolean expression in conjunctive normal form

$$\alpha = C_1 \wedge C_2 \wedge \dots \wedge C_m$$

for some $m \geq 1$, with

$$C_i = y_{i,1} \vee y_{i,2} \vee \dots \vee y_{i,p_i}$$

where $p_i \geq 1$, and $y_{i,j} \in \{x_k, \neg x_k | 1 \leq k \leq n\}$, for each $1 \leq i \leq m$, $1 \leq j \leq p_i$.

We build the P system

$$\Pi = (V, T, H, \mu, \omega_0, \omega_1, R)$$

where

- $V = \{a_i, t_i, f_i | 1 \leq i \leq n\} \cup \{r_i | 1 \leq i \leq m\} \cup \{W_i | 1 \leq i \leq m + 1\} \cup \{t\} \cup \{Z_i | 0 \leq i \leq n\}$
- $T = \{t\}$
- $H = \{0, 1\}$
- $\mu = [0[1]_1^0]_0^0$
- $\omega_0 = \lambda$
- $\omega_1 = a_1 a_2 \dots a_n Z_0$

while the set R contains the following rules:

1. $[a_i]_1^0 \rightarrow [t_i]_1^0 [f_i]_1^0, 1 \leq i \leq n$

We substitute one variable a_i in membrane 1 with two variables t_i and f_i . The membrane is divided in two membranes (the charge remains neutral for both membranes). In n steps we get all 2^n truth assignments for the n variables; each truth assignment is in a membrane labelled with 1.

2. $[Z_k \rightarrow Z_{k+1}]_1^0, 0 \leq k \leq n - 2$

3. $[Z_{n-1} \rightarrow Z_n W_1]_1^0$

4. $[Z_n]_1^0 \rightarrow []_1^+ \lambda$

We count n steps, the time needed to produce all the truth assignments. The step $n - 1$ introduces the symbol W_1 used in the next steps, while the step n changes the charge of the membranes labelled with 1.

5. $[t_i \rightarrow r_{h_1} \dots r_{h_j}]_1^+, 1 \leq i \leq n, 1 \leq h_j \leq m$ and x_i is in the clauses h_1, \dots, h_j

6. $[f_i \rightarrow r_{h_1} \dots r_{h_j}]_1^+, 1 \leq i \leq n, 1 \leq h_j \leq m$ and $\neg x_i$ is in the clauses h_1, \dots, h_j

In one step, each symbol t_i is replaced with some symbols r_{h_j} , indicating the clauses satisfied if we set $x_i = true$; each symbol f_i is replaced with some symbols, indicating the clauses satisfied if we set $x_i = false$ (i.e. $\neg x_i = true$).

After this step, we start the “VERIFY STEPS”: we have to verify if there is at least one membrane, labelled with 1, in which we get all symbols r_1, r_2, \dots, r_m (at least one symbol r_i for every i). In fact, this means that there is a truth assignment satisfying all the clauses.

7. $[r_1]_1^+ \rightarrow []_1^- r_1$

We first verify the presence of the symbol r_1 in membranes labelled with 1. Every membrane containing r_1 (i.e. every membrane with a truth assignment satisfying the first clause) sends this symbol outside (in membrane 0) and changes its charge from $+$ to $-$. The membranes not containing this symbol cannot further proceed their computation.

8. $r_1 \square_1^- \rightarrow [r_0]_1^+$

9. $[r_i \rightarrow r_{i-1}]_1^-, 1 \leq i \leq m$

10. $[W_i \rightarrow W_{i+1}]_1^-, 1 \leq i \leq m$

The membranes with negative charge can continue the computation. They increase the index of the symbols W_i (the counters the satisfied clauses) and decrease the index of the symbols r_i . The symbols r_1 in membrane 0 are sent back (as r_0) to the negative charged membranes labelled with 1, to change their charge from $-$ to $+$.

After applying the rules of type 8, 9, and 10 (in parallel) we can re-apply the rules of type 7 followed again by the application of the rules of type 8, 9 and 10. In $2m$ steps we verify the existence of a membrane containing all symbols r_i . It's easy to see that if a membrane does not contain a symbol r_i , the computation in that membrane halts in less than $2m$ steps. As a consequence, that membrane will not contain the symbol W_{m+1} . The membranes containing this symbol are the membranes containing all symbols r_i when the VERIFY STEPS started, i.e. the membranes with a truth assignment satisfying all the clauses.

11. $[W_{m+1}]_1^+ \rightarrow \square_1^+ t$

If a membrane labelled with 1 executes all $2m$ verify steps, it contains the symbol W_{m+1} . Thus we send out to membrane 0 the symbol t , indicating that there is a truth assignment satisfying all the clauses.

12. $[t]_0^0 \rightarrow \square_0^+ t$

A symbol t is sent outside membrane 0 and the charge of this membrane is changed from 0 to $+$. No further computation is possible. Thus, we have to look at the output of membrane 0 after $n + 2m + 5$ steps. If we get the symbol t , it means that there is a truth assignment satisfying α ; otherwise, the formula is not satisfiable. •

6 Conclusions

In papers like [7], [8] and [9], is pointed out the importance of consider variants of P systems which use features of biochemical inspiration, to obtain more “realistic” systems which can be implemented either in biochemical media or in electronic media.

We've presented here some results, to underline the fact that a feature of biochemical inspiration can be useful not only to replace other features of different inspiration (for example, of a formal language inspiration, like priority over evolution rules). These features can be used to obtain simpler and faster models too. We've illustrated this, by showing that variable thickness can be used to replace priority, while electrical charges can be used to obtain systems with a simple membrane structure (systems of depth two with two rules per membrane). Moreover, we've shown how to build P systems able to solve the Satisfiability problem (a well known NP complete problem) in linear time, using division for elementary membranes only.

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] S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999).
- [5] Gh Paun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (Febr. 1999), 139-152.
- [6] Gh. Paun, Computing with membranes, submitted, 1998 (see also TUCS Research Report No 208, November 1998 <http://www.tucs.fi>).
- [7] Gh. Paun, Computing with membranes - a variant: P systems with polarized membranes, *Auckland Univ.*, CDMTCS Report No. 098, 1999.
- [8] Gh. Paun, P systems with active membranes: attacking NP complete problems, submitted 1999 (see also CDMTCS Research report No. 102, 1999, *Auckland Univ.*, New Zeland, www.cs.auckland.ac.nz/CDMTCS)
- [9] Gh. Paun, G.. Rozenberg, A. Salomaa, Membrane computing with external output, submitted, 1999 (see also TUCS Research Report No. 218, December 1998, <http://www.tucs.fi>).
- [10] Gh. Paun, S. Yu, On synchronization in P systems, submitted, 1999 (see also CS Department TR No 539, *Univ. of Western Ontario*, London, Ontario, 1999, www.csd.uwo.ca/faculty/syu/TR539.html).
- [11] Gh. Paun, T. Yokomori, Membrane computing based on splicing, *proc. of 5th DIMACS Workshop on DNA Based Computers*, 1999, 213-227.
- [12] I. Petre, A normal form for P systems, *Bulletin of EATCS*, 67 (Febr. 1999), 165-172.
- [13] G. Rozenberg, A. Salomaa, eds. , *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.