Expressing Control Mechanisms in P systems by Rewriting Strategies *

Oana Andrei[1], Gabriel Ciobanu[2,3], and Dorel Lucanu[2]

[1] INRIA-LORIA, Nancy, France, Oana.Andrei@loria.fr
[2] "A.I.Cuza" University of Iaşi, Faculty of Computer Science
[3] Romanian Academy, Institute of Computer Science, Iaşi
{gabriel, dlucanu}@info.uaic.ro

Abstract. In this paper we provide a semantics for membrane systems given by rewriting strategies. We describe some control mechanisms in membrane computing defined over sets of rules rather than individual rules. Then we present the rewriting strategy formalism together with its operational semantics. We use strategies to define the semantics of the maximal parallel rewriting and priorities. Rewriting strategies are not enough to express the membrane computation involving promoters or inhibitors.

1 Control Mechanisms in Membrane Systems

In membrane systems (called also P systems) [8], the objects to evolve and the rules governing this evolution are chosen in a nondeterministic way. Moreover, this choice is exhaustive in the sense that no rule can be further applied in the same evolution step: this is the maximal parallel rewriting. A global clock is assumed, that is the same clock for all the regions of a membrane system. At each tick of this clock, a current configuration of the system is transformed into another one, and so defining a transition between the configurations of the system. A sequence of transitions is called a computation. A computation is halting if it reaches a halting configuration, one where no rules are applicable at all.

We have various control mechanisms in membrane systems. They are inspired by some biological entities. For instance, we have catalysts representing objects which appear on both left-hand and right-hand sides of a rule. The catalysts directly participate in rules (but are not modified by them), and they are counted as any other object such that the number of applications of a rule involving a catalyst is as large as the number of copies of the catalyst. They can be used to apply a certain rule in a sequential way, increasing the control of using the rule. Another controlling mechanism is given by activators, a formal representation of enzymes. An activator is related to a rule. The rules need activators to be applied, so the parallelism of each rule is limited to the number of its activators.

* This work has been supported by the research grant CEEX 47/2005, Romania

The activators can evolve in the same step (this is not possible for catalysts). Another control mechanism concerns the membrane permeability. Thus the membranes can be dissolved by the so-called action delta (the objects of a dissolved membrane remain in the region surrounding this membrane, while the rules are removed; the skin membrane cannot be dissolved), or made impermeable by the so-called action tau (no object can pass through such a membrane).

In this paper we refer to control mechanisms defined over sets of rules rather than individual rules. Such mechanisms are given by priorities, promoters and inhibitors. A priority relation among rules means that in each region we have a partial order relation on the set of rules, and a rule can be chosen (to process a multiset of objects) only if no rule of a higher priority is applicable in the same region. Promoters and inhibitors formalize the reaction enhancing and reaction prohibiting roles of various substances (molecules) present in cells. In membrane systems, promoters and inhibitors are represented as multisets of objects associated with given sets of rules. A rule from such a set of a given region can be used only if all the promoting objects are present, and none of the inhibiting objects is present in that region. From the generative point of view, there is a symmetry between the two ideas: systems with promoters are equal in power to systems with inhibitors, and they characterize the recursively enumerable sets of natural numbers. Membrane systems with promoters/inhibitors achieve universal computations without using other features of membrane systems. From a technical/efficiency point of view, there are differences: it is much easier to work with promoters, the needed constructions are simpler, and if we have enough promoters, then systems with only one membrane are already universal. An important feature of these universality results, both for promoters and inhibitors, is that their proofs are obtained without using any other standard feature in membrane computing [6].

Regarding the difference between promoters and catalysts, we can say that the catalysts directly participate in rules, and they are counted as objects required by rules, and the number of rules applied in parallel is as large as the number of catalysts. In the case of promoters, the presence of only one promoter makes it possible to use a rule involving that promoter as many times as possible, without any restriction. A restricted number of catalysts can help to control the number of application in parallel of the rules involving catalysts.

1.1 Membrane Systems with Priorities and Promoters

We present some examples of membrane systems implementing arithmetical operations on numbers represented by the numbers of objects. More details about other arithmetical operations on numbers represented by using unary and binary compact encodings are presented in [4]. In the examples presented here we emphasize the use of priorities and promoters as control mechanisms in membrane computing, presenting membrane systems with priorities and promoters for multiplication.

Figure 1 presents a membrane system Π_1 with priorities for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0.

$$\begin{aligned}
\Pi_1 &= (V, \mu, w_0, (R_0, \rho_0), 0),\\
V &= \{a, b, d, e, u, v\},\\
\mu &= [_0]_0,\\
w_0 &= a^n b^m u,\\
R_0 &= \{l_1 : bv \to dev,\ l_2 : av \to u,\ l_3 : eu \to dbu,\ l_4 : au \to v\},\\
\rho_0 &= \{l_1 > l_2,\ l_3 > l_4\}.
\end{aligned}$$

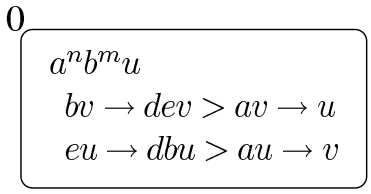

Fig. 1. Multiplication with priorities

In this system we use the priority relation between rules; for instance $bv \to dev$ has a higher priority than $av \to u$, meaning the second rule is applied only when the first one cannot be applied anymore. Initially only the rule $au \to v$ can be applied, generating a catalyst v which activates m times the rule $bv \to dev$. Then $av \to u$ consumes an a, and transform the catalyst v into a catalyst u. Now $eu \to dbu$ is applied m times, followed by another change of catalyst u into a catalyst v by consuming an a (this is done by the rule $au \to v$). The procedure is repeated until no object a is present within the membrane. It is easy to note that each time when one object a is consumed, then m objects d are generated.

Figure 2 presents a P system Π_2 with promoters for multiplication of n (objects a) by m (objects b), the result being the number of objects d in membrane 0. The object a is a promoter in the rule $b \to bd|_a$, i.e., this rule can only be applied in the presence of object a. The available m objects b are used in order to apply m times the rule $b \to bd|_a$ in parallel; based on the availability of a objects the rule $au \to u$ where u is a catalyst is applied in the same time and consumes an a. The procedure is repeated until no object a is present within the membrane. Note that each time when one object a is consumed, then m objects d are generated.

$$\begin{aligned}
\Pi_2 &= (V, \mu, w_0, R_0, 0),\\
V &= \{a, b, d, u\},\\
\mu &= [_0]_0,\\
w_0 &= a^n b^m u,\\
R_0 &= \{l_1 : b \to bd|_a,\ l_2 : au \to u\}.
\end{aligned}$$

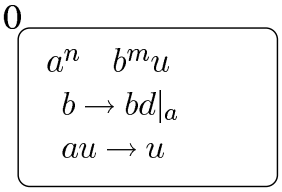

Fig. 2. Multiplication with promoters

The membrane systems for multiplication presented here do not require exponential space, and they do not require active membranes. Another interesting feature is that the computation may continue after reaching a certain result, and so the system acts as a P transducer [7]. Thus if initially there are n (objects a) and m (objects b), the system evolves and produces $n \cdot m$ objects d. Afterwards, the user can inject more objects a and the system continues the computation obtaining the same result as if the objects a are present from the beginning. For example, if the user wishes to compute $(n+k)\cdot m$, it is enough to inject k objects a at any point of the computation.
We have tested these examples by using the WebPS simulator described in [5]; it is available at http://psystems.ieat.ro.

2 Strategies and Tactics in Rewriting Systems

In general terms, a strategy is setting the objective(s) of a computation. On the other hand, tactics indicate how we are supposed to reach the objectives. In term rewriting systems, a strategy is an expression s involving rewriting rules and certain strategy operators. The objectives of s are *strategic transitions* $t \xrightarrow{s} t'$, where t and t' are terms. A tactic of a strategic transition $t \xrightarrow{s} t'$ is a rewriting sequence $t = t_0 \to \cdots \to t_n = t'$ denoted shortly by $t \rightsquigarrow t'$ which applies the rewriting rules according to a strategy s. We write $t \rightsquigarrow_n t'$ when we want to specify the length of the rewriting sequence.

We can think a rewriting strategy as an algorithm for defining a computation step induced by a set of rules. In particular, the rewriting rules are atomic strategies. For instance, computation of the arithmetical expressions involving associative and commutative $+$ and $*$, without parenthesis, could follow several pathways:

$$2+3*5 \to 2+15 \to 17$$
$$2+3*5 = 2+5*3 \to 2+15 \to 17$$
$$2+3*5 = 3*5+2 \to 15+2 \to 17$$
$$2+3*5 \to 5*5 \to 25$$
$$2+3*5 = 2+5*3 = 5+2*3 \to 5+6 \to 11$$
$$\ldots$$

Only some of them are correct according to the usual arithmetical rules (for the previous example, these are the pathways leading to 17). We aim to define a strategy `eval` such that $2 + 3 * 5 \xrightarrow{\texttt{eval}} 17$. In general, a rewriting strategy language consists of expressions s constructed with rewriting rules and strategy operators such that $expr \xrightarrow{s} expr'$. Here we use the strategy language defined in [10]. In this language, `eval` can be expressed as $\texttt{repeat}(\texttt{multiply} \leftarrow \texttt{add})$. This means that we apply multiplication repeatedly until it is not possible anymore, followed by addition.
In the sequel we present the strategy language.

Basic strategies. Each evolution rule $\ell : u \to v$ defines a strategy operator with the operational semantics given by the strategic transition $w \xrightarrow{\ell} w'$, where $w = u$, $w' = v$, and these equalities are modulo associativity, commutativity, and unity. The only rewriting tactic corresponding to such a strategic transition is $w \rightsquigarrow_1 w'$, i.e., the rewriting of length one defined by the rule. The uniqueness of the tactic is given by the fact the evolution rules have no variables.

Identity. We consider a strategy operator `id` with the operational semantics given by $w \xrightarrow{\texttt{id}} w$, where w is a configuration. The only rewriting tactic corresponding to such a strategic transition is $w \rightsquigarrow_0 w$, i.e., the rewriting of length zero.

Congruence. Each operation name (term constructor) defines a strategy operator whose parameter-strategies are applied to its arguments. Since in membrane systems we have only one operator, namely the associative and commutative concatenation, we use a specific strategy operator `mset` with a variable number of parameter-strategies. The operational semantics of the operator `mset` is

$$\frac{w_1 \xrightarrow{s_1} w_1' \quad \dots \quad w_n \xrightarrow{s_n} w_n'}{w_1 \cdots w_n \xrightarrow{\texttt{mset}(s_1,\dots,s_n)} w_1' \cdots w_n'}$$

Remark 1. The concatenation operator `__` has variable arity due to its associativity and unity laws. For instance, $aabbb$ can be written as $\texttt{\_\_}(aa, bbb)$, $\texttt{\_\_}(a, a, b, b, b)$, $\texttt{\_\_}(a, ab, bb)$, and so on. The notation $\texttt{\_\_}[\texttt{assoc comm id}: \varepsilon]$ intends to capture this feature together with commutativity of `__`. In order to avoid any confusion, we prefer to denote this variadic operator by `mset`. Therefore we use $\texttt{mset}(s_1, \dots, s_n)$ instead of $\texttt{\_\_}[\texttt{assoc comm id}: \varepsilon](s_1, \dots, s_n)$. The general case of congruence provided by operators with attributes is discussed in [3].

Example 1. Considering the rules in Figure 1, we have $abu \xrightarrow{\texttt{mset}(\ell_4,\text{id})} bv$ because $au \xrightarrow{\ell_4} v$, $b \xrightarrow{\text{id}} b$, $abu = aub$, and $vb = bv$.

If $w_i \rightsquigarrow w_i'$ is a rewriting tactic of $w_i \xrightarrow{s_i} w_i'$ for $i = 1, \dots, n$, then $w_1 \cdots w_n \rightsquigarrow w_1' \cdots w_n \rightsquigarrow \cdots \rightsquigarrow w_1' \cdots w_n'$ is a tactic of $w_1 \cdots w_n \xrightarrow{\texttt{mset}(s_1,\dots,s_n)} w_1' \cdots w_n'$. Since the concatenation is associative and commutative, the rewriting tactics can be combined in an arbitrary order.

Sequential composition. $s_1; s_2$ applies s_1 and, if it succeeds, then it applies s_2. The operational semantics of $s_1; s_2$ is given by

$$\frac{w \xrightarrow{s_1} w' \quad w' \xrightarrow{s_2} w''}{w \xrightarrow{s_1;s_2} w''}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$ and $w' \rightsquigarrow w''$ is a rewriting tactic of $w' \xrightarrow{s_2} w''$, then $w \rightsquigarrow w' \rightsquigarrow w''$ is a rewriting tactic of $w \xrightarrow{s_1;s_2} w''$.

Example 2. Since $abu \xrightarrow{\mathtt{mset}(\ell_4,\mathrm{id})} bv$ and $bv \xrightarrow{\ell_1} dev$, we have

$$abu \xrightarrow{\mathtt{mset}(\ell_4,\mathrm{id});\ell_1} dev.$$

Non-deterministic choice. $s_1 + s_2$ chooses between the strategies s_1 and s_2 such that the chosen strategy succeeds. The operational semantics of $s_1 + s_2$ is given by

$$\frac{w \xrightarrow{s_1} w'}{w \xrightarrow{s_1+s_2} w'} \qquad \frac{w \xrightarrow{s_2} w'}{w \xrightarrow{s_1+s_2} w'}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$ or $w \xrightarrow{s_2} w'$, then $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1+s_2} w'$.

Example 3. Considering the evolution rules in Figure 2, we have $abu \xrightarrow{\ell_1+\ell_2} bu$ or $abu \xrightarrow{\ell_1+\ell_2} abdu$.

Failure. We say that a strategy s *fails* on w iff there is no w' such that we cannot have $w \xrightarrow{s} w'$. We write $w \xrightarrow{s}\!\uparrow$. In other words, $w \xrightarrow{s}\!\uparrow$ means that for any w', $w \xrightarrow{s} w'$ has no rewriting tactics.

Deterministic choice. $s_1 +\!\!\!\!\cdot\, s_2$ chooses the left argument first; the second strategy is considered if the first strategy fails. The operational semantics of $s_1 +\!\!\!\!\cdot\, s_2$ is given by

$$\frac{w \xrightarrow{s_1} w'}{w \xrightarrow{s_1+\!\!\!\!\cdot\, s_2} w'} \qquad \frac{w \xrightarrow{s_1}\!\uparrow \quad w \xrightarrow{s_2} w'}{w \xrightarrow{s_1+\!\!\!\!\cdot\, s_2} w'}$$

If $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1} w'$, then $w \rightsquigarrow w'$ is a rewriting tactic of $w \xrightarrow{s_1+\!\!\!\!\cdot\, s_2} w'$. If for any w', $w \xrightarrow{s_1} w'$ has no rewriting tactics, then any rewriting tactic of $w \xrightarrow{s_2} w'$ is a a rewriting tactic of $w \xrightarrow{s_1+\!\!\!\!\cdot\, s_2} w'$.

Example 4. Using again the evolution rules in Figure 1, we have $abv \xrightarrow{\ell_1+\!\!\!\!\cdot\,\ell_2} adev$. We cannot deduce $abv \xrightarrow{\ell_1+\!\!\!\!\cdot\,\ell_2} bu$ because ℓ_1 succeeds on abv. On the other hand, ℓ_1 fails on $adev$ and so we have $adev \xrightarrow{\ell_1+\!\!\!\!\cdot\,\ell_2} deu$.

Strategy definition. A *strategy definition* is an expression $\varphi(z_1, \ldots, z_n) \stackrel{\text{def}}{=} s$, where any free variable in s belongs to $\{z_1, \ldots, z_n\}$, and φ is a strategy identifier. The operational semantics is given by

$$\frac{w \xrightarrow{s[z_1:=s_1,\ldots,z_n:=s_n]} w'}{w \xrightarrow{\varphi(s_1,\ldots,s_n)} w'} \text{ if } \varphi(z_1, \ldots, z_n) \stackrel{\text{def}}{=} s$$

where $s[z_1 := s_1, \ldots, z_n := s_n]$ is the strategy expression obtained from s by replacing the (free occurrences of the) variables z_i with s_i. Each rewriting tactic of $w \xrightarrow{s[z_1:=s_1,\ldots,z_n:=s_n]} w'$ is a rewriting tactic of $w \xrightarrow{\varphi(s_1,\ldots,s_n)} w'$.

Fixpoint operator. The fixpoint operator $\mu z(s)$ allows to define strategies that repeatedly apply a certain strategy s. For instance, the strategy `repeat`, which applies s as many times as possible, is defined as

$$\texttt{repeat}(s) \stackrel{\text{def}}{=} \mu z((s; z) \leftarrow\!\!+ \mathsf{id})$$

The operational semantics of $\mu z(s)$ is given by

$$\frac{w \xrightarrow{s[z:=\mu z(s)]} w'}{w \xrightarrow{\mu z(s)} w'}$$

The fixpoint operator μ binds any occurrence of variable z in strategy s.
Each rewriting tactic of $w \xrightarrow{s[z:=\mu z(s)]} w'$ is a rewriting tactic of $w \xrightarrow{\mu z(s)} w'$.

3 Strategy Semantics of Control Mechanisms

In membrane systems, a computation step $w \Rrightarrow w'$ can be presented as a transition from a configuration to another configuration according to a control mechanism involving priorities or promoters. When we refer to a sequential implementation for membrane computing, such a computation is translated in sequential rewritings. Such a sequential implementation based on rewritings is presented in [1]. In this paper, it is natural to think about computations as objectives provided by strategies, and sequential implementations as tactics of these strategies.

We investigate whether we can find a strategy s such that $w \Rrightarrow w'$ iff $w \xrightarrow{s} w'$. We describe the strategies for expressing control mechanisms defined over sets of rules rather than individual rules. Such mechanisms are given by priorities, promoters and inhibitors. However the most important computing engine is represented by the maximal parallel rewriting. We give a strategic semantics for maximal parallel rewriting, as well as for maximal parallel rewriting with priorities between rules. However we find that a more powerful mechanism than strategies is needed to provide semantics for maximal rewriting with promoters or inhibitors. A useful encoding of the rules can solve this problem, and finally we can provide the semantics for maximal rewriting of membrane systems involving promoters/inhibitors. The encoding can be used in a uniform way to

provide the strategy semantics for simple maximal rewriting, maximal rewriting of membrane systems with priorities, and maximal rewriting of systems with promoters/inhibitors.

3.1 Strategic Semantics of Maximal Parallel Rewriting

Let R be a set of evolution rules and w a multiset of objects. If $\ell : u \to v$ an evolution rule in R, then w is *ℓ-irreducible* if we cannot infer $w \xrightarrow{\mathtt{mset}(\ell,\mathrm{id})} w'$. Moreover, w is *R-irreducible* if w is ℓ-irreducible for all $\ell : u \to v \in R$. In other words, w is ℓ-irreducible iff there is no w' such that $w \to w'$ applying the rule labelled by ℓ. We say that w *maximal parallel rewrites* in w', write $w \Rrightarrow_R w'$, iff $w = u_1 \cdots u_n z$, $w' = v_1 \cdots v_n z$, $\ell_i : u_i \to v_i$ is a rule in R for $i = 1, \ldots, n$, $n > 0$, and z is R-irreducible.

Given a set of evolution rules $R = \{\ell_i : u_i \to v_i \mid 1 \le i \le n\}$, we define a strategy

$$\begin{aligned} &mpr \stackrel{\text{def}}{=} \mu x(s_1 + \cdots + s_n) \\ &\text{where } s_i = \mathtt{mset}(\ell_i, x \leftarrowtail \mathrm{id}), \text{ for } i = 1, \ldots, n. \end{aligned}$$

Since the definition of the strategy mpr is depending on R, we prefer to write it in an equivalent form

$$mpr(R) \stackrel{\text{def}}{=} \mathtt{mset}(\ell_1, mpr(R) \leftarrowtail \mathrm{id}) + \cdots + \mathtt{mset}(\ell_n, mpr(R) \leftarrowtail \mathrm{id})$$

If $R = \emptyset$, then $w \xrightarrow{mpr(\emptyset)} \uparrow$ for any w.

Theorem 1. *Given a set R of evolution rules, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{mpr(R)} w'.$$

Proof. We first assume that $w \Rrightarrow_R w'$. We have $w = u_{i_1} \cdots u_{i_k} z$, $w' = v_{i_1} \cdots v_{i_k} z$, rules $\ell_{i_j} : u_{i_j} \to v_{i_j}$ from $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$ for $j = 1, \ldots, k$, $k > 0$, and z is R-irreducible. We show $w \xrightarrow{mpr(R)} w'$ by induction on k.
If $k = 1$, then the proof is:

$$\dfrac{\dfrac{\dfrac{\dfrac{z \xrightarrow{mpr(R)} \uparrow \qquad z \xrightarrow{\mathrm{id}} z}{z \xrightarrow{mpr(R) \leftarrowtail \mathrm{id}} z} \qquad u_i \xrightarrow{\ell_i} v_i}{u_i z \xrightarrow{s_i} v_i z}}{u_i z \xrightarrow{s_1 + \cdots + s_n} v_i z}}{u_i z \xrightarrow{mpr(R)} v_i z}$$

where $i = i_1$.
If $k > 1$, then $u_{i_k} z \xrightarrow{mpr(R)} v_{i_k} z$ as above, and $u_1 \ldots u_{i_{k-1}} \xrightarrow{mpr} v_1 \ldots v_{i_{k-1}}$ by inductive hypothesis. We get $w \xrightarrow{mpr(R)} w'$ by the definition of the fixpoint operator, and by the fact that the concatenation in the left hand side does not produce new reducible configurations.

Conversely, if $w \xrightarrow{mpr(R)} w'$ then we show that $w \Rrightarrow_R w'$ by induction on the depth of the proof tree. There are $\ell_i : u_i \to v_i$ in R, w_i and w_i' such that $w = u_i w_i$, $w' = v_i w_i'$, and $w_i \xrightarrow{mpr(R)} w_i'$, by definition of $mpr(R)$. We have $w_i \Rrightarrow_R w_i'$ by inductive hypothesis, and we get $w = u_i w_i \Rrightarrow_R v_i w_i' = w'$ by the definition of $\Rrightarrow$.

Example 5. If R consists of the rules $\ell_1 : ab \to c$ and $\ell_2 : bb \to d$, then the inference tree for $aabbb \xrightarrow{mpr} cda$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a \xrightarrow{mpr(R)} \uparrow \qquad a \xrightarrow{\mathrm{id}} a}{a \xrightarrow{mpr(R) +\!\!+ \mathrm{id}} a} \qquad bb \xrightarrow{\ell_2} d}{abb \xrightarrow{s_2} da}}{abb \xrightarrow{s_1 + s_2} da}}{abb \xrightarrow{mpr(R)} da}}{abb \xrightarrow{mpr(R) +\!\!+ \mathrm{id}} da} \qquad ab \xrightarrow{\ell_1} c}{aabbb \xrightarrow{s_1} cda}}{aabbb \xrightarrow{s_1 + s_2} cda}}{aabbb \xrightarrow{mpr(R)} cda}$$

3.2 Maximal Parallel Rewriting with Priorities

Let R be a set of evolution rules together with a partial order $\succ$. If $\ell \succ \ell'$, then we say that ℓ has a greater priority than ℓ'. An evolution rule is applied in an evolution step only if no rule of a higher priority can be applied.

Definition 1. *Let R be a set of evolution rules together with a priority relation $\succ$. We say that w* maximally parallel rewrites *in w' w.r.t. R, and write $w \Rrightarrow_R w'$, iff $w \Rrightarrow_{max(R,w)} w'$, where $max(R, w)$ is the set of evolution rules in R of highest priority which are applicable to w.*

Note that $max(R, w)$ is a discrete partial ordered set, and consequently $w \Rrightarrow_{max(R,w)} w'$ is defined as in 3.1. However, we cannot apply the strategy $mpr(max(R, w))$ because it depends on configuration w. Usually a strategy is independent of the configuration.

Definition 2. *Let R be a set of evolution rules together with a priority relation $\succ$ such that $\{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$ is the subset of the rules with maximal priority. Then the strategy $pri(R)$ is defined as follows:*

$$\begin{aligned} &pri(R) \stackrel{def}{=} s_1 + \cdots + s_n, \\ &s_i = \mathtt{mset}(\ell_i, pri(filter(R, \ell_i)) \leftarrow id) \leftarrow pri(R \setminus \{l_i\}) \\ &\text{for } i = 1, \ldots, n, \end{aligned}$$

where $filter(R, \ell_i)$ is obtained from R by removing the rules having lower priority than ℓ_i. If $R = \emptyset$, then $w \xrightarrow{pri(\emptyset)} \uparrow$.

Example 6. Let us suppose that R consists of $\ell_1 : a \to c \succ \ell_2 : b \to d$.
We have:

$$pri(R) \stackrel{\text{def}}{=} s_1 \quad (\ell_1 \text{ is the only maximal element in } R)$$
$$s_1 = \texttt{mset}(\ell_1, pri(\ell_1) \leftarrow\!\!+ \text{id}) \leftarrow\!\!+ pri(\ell_2)$$
$$pri(\ell_1) \stackrel{\text{def}}{=} \texttt{mset}(\ell_1, pri(\ell_1) \leftarrow\!\!+ \text{id})$$
$$pri(\ell_2) \stackrel{\text{def}}{=} \texttt{mset}(\ell_2, pri(\ell_2) \leftarrow\!\!+ \text{id})$$

The inference tree for $aab \Rrightarrow ccb$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{b \xrightarrow{pri(\ell_1)} \uparrow \qquad b \xrightarrow{\text{id}} b}{b \xrightarrow{pri(\ell_1) \leftarrow\!\!+ \text{id}} b} \qquad a \xrightarrow{\ell_1} c}{ab \xrightarrow{\texttt{mset}(\ell_1, pri(\ell_1) \leftarrow\!\!+ \text{id})} cb}}{ab \xrightarrow{pri(\ell_1)} cb}}{ab \xrightarrow{pri(\ell_1) \leftarrow\!\!+ \text{id}} cb} \qquad a \xrightarrow{\ell_1} c}{aab \xrightarrow{\texttt{mset}(\ell_1, pri(\ell_1) \leftarrow\!\!+ \text{id})} ccb}}{aab \xrightarrow{\texttt{mset}(\ell_1, pri(filter(R, \ell_1)) \leftarrow\!\!+ \text{id})} ccb}}{aab \xrightarrow{pri(R)} ccb}$$

Theorem 2. *Given a set R of evolution rules together with a priority relation $\succ$,*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{pri(R)} w'.$$

Proof. (Sketch) The main idea is similar to that in the proof of Theorem 1. The correct handling of the priorities is assured by the following facts:

- only rules with maximal priority are applied,
- once a rule is applied, all the rules having smaller priorities are removed from the current set of rules by the operator $filter$, and
- if a rule with a maximal priority cannot be applied, then it is removed.

3.3 Maximal Parallel Rewriting with Promoters

An *evolution rule with promoter* is a rewrite rule of the form $\ell : u \to v|_p$, where the promoter p does not necessarily occur in u. Such a rule can be applied in an evolution step $w \Rrightarrow w'$ only if w contains both u and p. A promoter cannot evolve by the same rule it promotes, but it can evolve by another rule. The problem we try to solve in this section is if the rules with promoters can be implemented with strategies. Starting from the above intuitive definition, we may be tempted to implement a rule with promoter by the following strategy:

$$\texttt{mset}(\text{id}(p), \ell, \text{id})$$

where $\text{id}(p)$ is the strategy corresponding to the rule $\text{id}(p) : p \to p$; the role of this rule is to test the presence of the promoter p. Note that a single occurrence of the promoter can be used by more than one rule or even the promoter can

be consumed by some other rule, and the presence of the promoters makes it possible to use a rule from the associated set as many times as possible, without any restriction. For instance, let R consist of the following rules with promoters: $\ell_1 : aq \to c|_p$ and $\ell_2 : bp \to d|_q$. Consider s a strategy applying the rules R over $abpq$. If s applies first ℓ_1, then the information that we initially had the promoter p is lost and it does not know that ℓ_2 can be applied. If s applies first ℓ_2, then the information that we initially had the promoter q is lost and it does not know that ℓ_1 can be applied. Therefore we claim that there is no strategy expressed in terms of rules R and strategy operators which implements the maximal rewriting with promoters. A more powerful mechanism is needed. We show that an encoding together with the strategies over the translated rules are enough for implementing the maximal rewriting with promoters.

Given a set R of evolution rules with or without promoters, we construct a set $\widehat{R}$ of rewrite rules and a strategy $prom(\widehat{R})$ such that $w \Rrightarrow_R w'$ iff $w \xrightarrow{prom(\widehat{R})} w'$. Let $pset(R, w)$ denote the set of promoters occurring in w w.r.t. the set of rules R. The set $\widehat{R}$ consists of the following rules:

- $\texttt{compute} : w \to (w, pset(R, w))$,
- $\texttt{forget} : w'(w, s) \to w'w$, together with
- a rule $\hat{\ell} : w'(wu, s) \to w'v(w, s)$ for each rule $\ell : u \to v$ without promoter, and
- a rule $\hat{\ell} : w'(wu, ps) \to w'v(w, ps)$ for each rule $\ell : u \to v|_p$ with promoter,

where w', w range over configurations, and s range over sets of promoters.
The rule `compute` stores the set of promoters occurring in w as the second component of the pair and it remains unchanged during the application of the evolution rules. This information is used by the rules with promoters: such a rule is applied only if its promoter is present in the second component. Note that the processed part lies in the front of the pair and it is not affected by the next evolution rules applied in the current step; the evolution rules consumes only from the first component of the pair.

Definition 3. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $prom(\widehat{R})$ *is*

$$prom(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

For the example given above, $\widehat{R}$ consists of `compute`, `forget`, together with $\hat{\ell}_1 : w'(aqw, ps) \to w'c(w, ps)$ and $\hat{\ell}_2 : w'(bpw, qs) \to w'd(w, ps)$.

The inference tree of $abpq \xrightarrow{prom(\widehat{R})} cd$ is obtained as follows:
T_1:

$$\frac{(abpq, pq) \xrightarrow{\hat{\ell}_2} c(bp, pq)}{(abpq, pq) \xrightarrow{\hat{\ell}_1 + \hat{\ell}_2} c(bp, pq)}$$

T_2:

$$\dfrac{\dfrac{c(bp,pq) \xrightarrow{\hat{\ell}_2} cd(\varepsilon,pq)}{c(bp,pq) \xrightarrow{\hat{\ell}_1+\hat{\ell}_2} cd(\varepsilon,pq)} \qquad cd(\varepsilon,pq) \xrightarrow{\hat{\ell}_1+\hat{\ell}_2} \uparrow}{c(bp,pq) \xrightarrow{\mathtt{repeat}(\hat{\ell}_1+\hat{\ell}_2)} cd(\varepsilon,pq)}$$

T_3:

$$\dfrac{abpq \xrightarrow{\mathtt{compute}} (abpq,pq) \qquad \dfrac{\dfrac{T_1 \qquad T_2}{(abpq,pq) \xrightarrow{(\hat{\ell}_1+\hat{\ell}_2);\mathtt{repeat}(\hat{\ell}_1+\hat{\ell}_2)} cd(\varepsilon,pq)}}{(abpq,pq) \xrightarrow{\mathtt{repeat}(\hat{\ell}_1+\hat{\ell}_2)} cd(\varepsilon,pq)}}{abpq \xrightarrow{\mathtt{compute};\mathtt{repeat}(\hat{\ell}_1+\hat{\ell}_2)} cd(\varepsilon,pq)}$$

Finally,

$$\dfrac{T_3 \qquad cd(\varepsilon,pq) \xrightarrow{\mathtt{forget}} cd}{abpq \xrightarrow{prom(\widehat{R})} cd}$$

Theorem 3. *Given a set R of evolution rules with promoters, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{prom(\widehat{R})} w'.$$

Proof. (Sketch) $w \Rrightarrow_R w'$ iff $(w, pset(R,w)) \xrightarrow{\mathtt{repeat}(\hat{\ell}_1+\cdots+\hat{\ell}_n)} (w', pset(R,w))$. If $w \Rrightarrow_R w'$, then it follows that $w = u_{i_1}\cdots u_{i_k}z$, $w' = v_{i_1}\cdots v_{i_k}z$, either $\ell_{i_j} : u_{i_j} \to v_{i_j}$ or $\ell_{i_j} : u_{i_j} \to v_{i_j}|_p$ is a rule in $R = \{\ell_i : u_i \to v_i \mid i = 1,\ldots,n\}$ for $j = 1,\ldots,k$, $k > 0$, and z is R-irreducible. If $\ell_{i_j} : u_{i_j} \to v_{i_j}|_p$ is a rule involving a promoter p, then p occurs in w, and it belongs to $pset(R,w)$. We prove that $w \xrightarrow{prom(\widehat{R})} w'$ by induction on k

Conversely, if $(w, pset(w)) \xrightarrow{\mathtt{repeat}(\hat{\ell}_1+\cdots+\hat{\ell}_n)} (w', pset(R,w))$, then we prove $w \Rrightarrow_R w'$ by induction on the depth of the inference tree.

3.4 Maximal Parallel Rewriting with Inhibitors

An *evolution rule with inhibitor* is a rewrite rule of the form $\ell : u \to v|_{\neg p}$. Such a rule can be applied in an evolution step $w \Rrightarrow w'$ only if the inhibitor is not present in w.

We proceed in a similar way to that of promoters but taking in the rule `compute` the complementary set of inhibitors occurring in w w.r.t. the whole set of objects A, denoted by $iset(A,w)$ instead of $pset(R,w)$, and considering rules with inhibitors instead of rules with promoters:

- `compute` : $w \to (w, iset(A,w))$,
- `forget` : $w'(w,s) \to w'w$, together with
- a rule $\hat{\ell} : w'(wu,s) \to w'v(w,s)$ for each rule $\ell : u \to v$ without inhibitor, and
- a rule $\hat{\ell} : w'(wu,ps) \to w'v(w,ps)$ for each rule $\ell : u \to v|_{\neg p}$ with inhibitor,

where w', w range over configurations, and s range over sets of inhibitors.

Definition 4. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $inhib(\widehat{R})$ *is*

$$inhib(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

Theorem 4. *Given a set* R *of evolution rules with inhibitors, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{inhib(\widehat{R})} w'.$$

3.5 Maximal Parallel Rewriting with Promoters and Inhibitors

When we have rules involving both promotors and inhibitors, we encode a configuration by a triple $(w, pset(R, w), iset(A, w))$ in order to have information about both promotors and inhibitors. The set $\widehat{R}$ includes the following rules:

- $\texttt{compute} : w \to (w, pset(R, w), iset(A, w))$,
- $\texttt{forget} : w'(w, s, s') \to w'w$, together with
- a rule $\hat{\ell} : w'(wu, s, s') \to w'v(w, s, s')$ for each rule $\ell : u \to v$ in R without promoter or inhibitor,
- a rule $\hat{\ell} : w'(wu, ps, s') \to w'v(w, ps, s')$ for each rule $\ell : u \to v|_p$ in R with promoter, and
- a rule $\hat{\ell} : w'(wu, s, ps') \to w'v(w, s, ps')$ for each rule $\ell : u \to v|_{\neg p}$ in R with inhibitor.

This encoding is general, and it can be used even if one or both sets of promotors and inhibitors are empty.

Definition 5. *We suppose that* $R = \{\ell_i : u_i \to v_i \mid i = 1, \ldots, n\}$*, and let* $\widehat{R}$ *be computed as above. Then the strategy* $prominhib(\widehat{R})$ *is*

$$prominhib(\widehat{R}) \stackrel{def}{=} \texttt{compute}; \texttt{repeat}(\hat{\ell}_1 + \cdots + \hat{\ell}_n); \texttt{forget}$$

Theorem 5. *Given a set* R *of rules involving eventually promoters and inhibitors, then*

$$w \Rrightarrow_R w' \text{ iff } w \xrightarrow{prominhib(\widehat{R})} w'.$$

4 Conclusion

The main contribution of the paper is given by the novel use of strategies and tactics in defining the operational semantics of membrane systems. Another approach in defining the operational semantics of membrane systems is presented in [2]. An important feature of these systems is that many parallel rules are applied in a single step, and there is a relationship between rules and resources. We use strategies to describe the control mechanisms in membrane systems defined

by sets of rules rather than individual rules, namely maximal parallel rewriting, priorities, promoters and inhibitors. We do not know a similar approach relating strategies to membrane computing.

We adapt the strategy language presented in [9], taking into consideration that:

- the configurations are defined by multisets of objects, and so it is not necessary to use all the strategy operators defined in [9],
- we extend the strategy congruence operators for associative and commutative operations,
- control mechanisms involving promoters and inhibitors require an encoding of the configurations and more complex operations over them.

One of the challenges in membrane community is to derive a programming paradigm inspired by membrane computing. Starting from some examples for arithmetical operations, we have remarked the importance of certain control mechanisms which can play in membrane paradigm a similar role to `if-then-else`, `for` and `while` instructions in imperative programming, for instance. In this paper we investigate the possibility to define a membrane programming paradigm by using an extended strategy language. Despite the fact that maximal parallel rewriting and priorities are expressible by strategies, the control mechanisms involving promoters and inhibitors require an encoding and additional operations. We think that for these control mechanisms we need more powerful specification tools.

References

1. O. Andrei, G. Ciobanu, D. Lucanu. Executable Specifications of the P Systems. In *Membrane Computing. WMC5*, Lecture Notes in Computer Science vol.3365, Springer, 127–146, 2005.
2. O. Andrei, G. Ciobanu, D. Lucanu. A Structural Operational Semantics of the P Systems, In *Membrane Computing. WMC6*, Lecture Notes in Computer Science vol.3850, Springer, 32–49, 2006.
3. O. Andrei, G. Ciobanu, D. Lucanu. *Strategies and Tactics in Operational Semantics*, "A.I.Cuza" University, Faculty of Computer Science Tech.Rep. TR06-01, 2006.
4. C. Bonchiş, G. Ciobanu, C. Izbaşa. Encodings and Arithmetic Operations in Membrane Computing. In *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science vol.3959, Springer, 618–627, 2006.
5. C. Bonchiş, G. Ciobanu, C. Izbaşa, D. Petcu. A Web-based P systems simulator and its parallelization. In *Unconventional Computing*. Lecture Notes in Computer Science vol.3699, Springer, 58–69, 2005.
6. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg. Membrane systems with promoters/inhibitors. *Acta Informatica*, vol.38, 695–720, 2002.
7. G. Ciobanu, Gh. Păun, Gh. Ştefănescu. P Transducers, *New Generation Computing* vol.24, 1–28, 2006.
8. Gh. Păun. *Membrane Computing. An Introduction*. Springer, 2002.
9. E. Visser. A Survey of Strategies in Rule-Based Program Transformation Systems. *Journal of Symbolic Computation* vol.40, 831–873, 2005.
10. E. Visser, Z-A. Benaissa, A. Tolmach. Building program optimizers with rewriting strategies. *ACM SIGPLAN Notices*, vol.34, 13–26, 1999.