P Systems with Symport/Antiport of Rules

Matteo CAVALIERE
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
E-mail: matteo.cavaliere@estudiants.urv.es

Daniela GENOVA
Department of Mathematics
University of South Florida
Tampa, FL 33620, USA
E-mail: genova@helios.acomp.usf.edu

Abstract. Moving "instructions" instead of "data", using transport mechanisms inspired by biology – this could represent, shortly, the basic idea of the computing device presented in this paper. Specifically, we propose a new class of P systems that use, at the same time, evolution rules and symport/antiport rules. The idea of this kind of systems is simple: during a computation symbol-objects (the "data") evolve using evolution rules but they cannot be moved; on the other hand, the evolution rules (the "instructions") can be moved across the membranes using classical symport/antiport rules. We present different results using different combinations between the power of the evolution rules (catalytic, non-cooperative rules) and the weight of the symport/antiport rules. In particular, we show that, using non-cooperative rules and antiports of unbounded weight is possible to obtain at least the Parikh set of ET0L languages. On the other hand, using catalytic rules (one catalyst) and antiports of weight 2, the system becomes universal. Several open problems are also presented.

1 Introduction

We introduce a new model of P system that is obtained joining in an "exotic" way two well known models of P systems: the one with symbol-objects and the one with symport/antiport (in what follows we suppose the reader familiar with the basic concepts of membrane computing, as for example presented in [8]).

In a previous paper, [3], a model called evolution-communication P system has been introduced and studied.

There, inspired by what happens in biology, the computation has been divided in two phases: evolution of symbol-objects (consisting in the application of simple evolution rules with no target indications) and communication between the regions of the system (consisting in the application of symport/antiport rules).

On the other hand in, [9] Gh. Păun suggested a model of P system where the rules are moved across the membranes rather than the objects processed by these rules. In the

model presented in [9] the migration of the simple evolution rules is governed by metarules existing in all the regions.

We believe that, following the philosophy of the evolution-communication model, another way to move rules across the membranes is by using the most typical transport mechanism of P systems: symport/antiport rules. Therefore a P system with symport/antiport of rules is a P system with simple evolution rules (used to evolve the symbol-objects in the classical way but without communication targets) and symport/antiport rules that are used, during the computation, to move the evolution rules across the membranes. The output of a computation is defined as standard in the P system area: it is the number of objects produced in the output region at the end of an halting computation (i.e., when no rules can be applied anymore in any region).

In this paper we give the formal definition of this new model, some examples that describe its working and the proofs of results that we can consider "preliminary".

In particular we prove that, when using non-cooperative evolution rules, our systems can generate (at least) the Parikh set of ET0L languages using antiports of unbounded weight. We obtain the same result with antiports of bounded weight by using priorities among the transport rules.

On the other hand, if we use non-cooperative evolution rules, antiports of weight two and no priorities among the transport rules, then, our systems can generate (at least) the Parikh set of the family of languages generated by Indian parallel grammars.

When using catalytic rules (and in particular one catalyst) then P systems with symport/antiport of rules become universal: they can generate the Parikh set of the family of recursively enumerable languages, using antiports of weight 2.

We would like to conclude this introduction with some (maybe) "philosophical" considerations: moving rules instead of objects is, somehow, a new way of computing where, instead of moving data, we move the instructions that act on such data (just opposite to the classical approach used, for example, in the area of distributed computing also linked with P systems area in several papers; see for instance [6]).

It would be interesting to compare this "non-standard" approach of computing with the classical ones, looking to possible advantages and disadvantages; we prefer to leave this topic for the interested reader.

2 Definition

We formally define the new variant of P systems as systems that use evolution rules as defined in [8], chapter 3 (but without communication targets, or, equivalently, with all the communication targets fixed as "here"), and symport/antiport rules, as defined in [8], chapter 4, used here to move evolution rules across the membranes of the system. For simplicity, we often call the evolution rules without communication targets *simple evolution rules* (or *simple catalytic rules*). It should be noticed that, differently from classical symport/antiport defined over *multiset* of elements, here such kind of rules are used over *set* of elements.

Definition 2.1 *A* P system with symport/antiport of rules *(in short,* an CR P system*, with CR coming from "communication of rules"), of degree* $m \geq 1$*, is defined as*

$$\Pi = (O, R, l, \mu, w_1, w_2, \cdots, w_m, R_1, \cdots, R_m, R'_1, \cdots, R'_m, i_o),$$

where:

- *O is the alphabet of objects;*
- *R is a finite set of simple evolution rules;*
- *l is a injective labeling of the rules in R; we denote $L = \{l(r) \mid r \in R\}$;*
- *μ is a membrane structure with m membranes (and hence m regions) injectively labeled with $1, 2, \cdots, m$;*
- *w_i are strings which represent multisets over O associated with the regions $1, 2, \cdots, m$ of μ;*
- *$R_i \subseteq R$, $1 \leq i \leq m$, are finite sets of simple evolution rules over O; R_i is associated with the region i of μ; a simple evolution rule is of the form $u \to v$, where u and v are strings over the alphabet O;*
- *R'_i, $1 \leq i \leq m$, are finite sets of symport/antiport rules; R'_i is associated with the membrane i of μ. A symport rule is of the kind (x, in) or (y, out), while an antiport rule is of the kind $(x, in; y, out)$, where x, y are strings that represent sets of elements in L.*
- *$i_o \in \{0, 1, 2, \cdots, m\}$ is the output region; if $i_o = 0$, then it is the environment, otherwise i_o is the label of an elementary membrane of μ.*

In a P system with symport/antiport of rules objects never pass through membranes, but they can change to other objects using simple evolution rules. R is the set of all possible simple evolution rules that the system can use. Each rule in R is labeled with an unique label. Symport/antiport rules constructed over the set of labels associated to the rules are used to move, during the computation, the simple evolution rules across the membranes.

A configuration is described using the m-tuple of multisets of objects together with the simple evolution rules, present in the m regions of the system. To each region is associated a finite number of objects and a finite number of simple evolution rules; to each membrane is associated a finite set of symport/antiport rules.

In every region the rules are present in the *set sense*, i.e., we cannot have more than one copy of a rule in one region, unlike objects where multiplicity is essential.

A transition between two configurations is governed by the *mixed application* of the evolution rules and of the symport/antiport rules. All objects which can evolve through evolution rules should evolve and all the evolution rules that can be moved by symport/antiport rules should be moved. However, if an evolution rule acts on an object in a region i, then it cannot be moved in the same step, using the symport/antiport rules associated to membrane i, and viceversa. There is no difference between evolution rules and symport/antiport rules: they are chosen and applied in the non-deterministic maximally parallel way.

The evolution rules work on the symbol-objects in the standard way; symport/antiport rules work in a standard way except the fact that they move rules (using their labels) and not objects. If a symport rule (x, in) associated to membrane i is applied, then the evolution rules represented by the string x pass into the membrane i from the region surrounding the membrane i. If the symport (x, out) is applied to membrane i, then the evolution rules represented by string x move up from such membrane to the region (or to the environment) that surrounds the membrane i.

Finally, if the antiport rule $(x, in; y, out)$ is applied to membrane i, then the evolution rules represented by x pass into region i from the region surrounding it, while, at the same time the evolution rules represented by y move in the opposite direction.

For simplicity, in what follows, we use a *short notation*: when we say $x = \langle S \rangle$, where S is a set, then x is the string representing S (i.e, x is a string obtained by concatenating, in an arbitrary order, elements of S where each element is taken exactly once).

A sequence of transitions is called a *computation* and a computation is considered *successful* (or halting) if it starts in the initial configuration and ends in a halting configuration (a configuration where no evolution rule and no symport/antiport rule can be applied, in any region).

The result of a successful computation is the number of objects present at the end of the computation in a designated region (output region). This way to have a computation will be called the *mixed approach*.

We use the following notation

$$PsCRP_m(i, j, \alpha), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by CR P systems with at most m membranes, using symport rules of weight at most i, antiport rules of weight at most j (as usually, $m = *, i, j = *$ if such a number is unbounded) and simple evolution rules that can be cooperative (coo), non-cooperative ($ncoo$), or catalytic (cat_k), using at most k catalysts.

3 How CR P Systems Work: Two Examples

We show the working of a CRP system using two simple examples. In particular we show how to construct a CRP system generating the Parikh set of the (non-semilinear) language $\{a^{2^n} \mid n \in N\}$. In Example 3.1, antiports of weight 1 are used. In Example 3.2, symports of weight 2 are used.

3.1 Using Antiport of Rules

We construct the following system, of degree 2, defined as

$$\Pi = (O, R, l, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, 2),$$

where:

- $O = \{A, a\}$;
- $R = \{A \to a, A \to AA\}$;
- l is a injective labeling of the rules in R;
  we use $l_1 = l(A \to a), l_2 = l(A \to AA); L = \{l_1, l_2\}$;
- $\mu = [_1[_2]_2]_1$;
- $w_1 = \emptyset, w_2 = A$;
- $R_1 = \{A \to a\}$;
- $R_2 = \{A \to AA\}$;

- $R'_1 = \emptyset$;
- $R'_2 = \{(l_1, in; l_2, out)\}$.

The system Π generates the set of natural number $\{a^{2^n} \mid n \geq 0\}$. At the beginning of the computation only the symbol object A is present in region 2. Moreover the rule $A \to AA$ with label l_2 is present in region 2 and the rule $A \to a$ with label l_1 is present in region 1. The rule $A \to AA$ is applied an arbitrary number of times in region 2. Some time, using the antiport $(l_1, in; l_2, out)$ (associated to membrane 2) we move the rule $A \to a$ from region 1 to region 2 and the rule $A \to AA$ from region 1 to region 2. Once the rule $A \to a$ is inside region 2 then all copies of A are transformed into $a's$ and no rule can be applied anymore in any region (then the computation halts and we get the result in region 2).

3.2 Using Symport of Rules

We construct the following system, of degree 2, defined as

$$\Pi = (O, R, l, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, 2),$$

where:

- $O = \{A, a, C\}$;
- $R = \{A \to a, A \to AA, C \to C\}$;
- l is a injective labeling of the rules in R;
  we use $l_1 = l(A \to a), l_2 = l(A \to AA), l_3 = (C \to C); L = \{l_1, l_2, l_3\}$;
- $\mu = [_1[_2]_2]_1$;
- $w_1 = \emptyset, w_2 = A$;
- $R_1 = \{A \to a\}$;
- $R_2 = \{A \to AA, C \to C\}$;
- $R'_1 = \emptyset$;
- $R'_2 = \{(l_3 l_2, out), (l_3 l_1, in)\}$.

At the beginning of the computation only the symbol-object A is present in region 2. The rules $A \to AA$ and $C \to C$ are present in region 2. The rule $A \to AA$ is applied an arbitrary number of times in region 2. Some time, using the symport $(l_3 l_2, out) \in R'_2$ the rules $A \to AA$ and $C \to C$ move from region 2 to region 1. After such step in region 1 there are the rules $A \to AA, C \to C, A \to a$. Because the symport rules are applied in a maximally parallel way, then the symport $(l_3 l_1, in) \in R'_2$ is applied and the rules $A \to a$ and $C \to C$ move from region 1 to region 2 (notice that, because of the dummy rule $C \to C$, this symport can be applied only after the rule $A \to AA$ exits from region 2). Therefore when the rule $A \to a$ is in region 2 all the symbol-objects A are changed to a and the computation halts (we get the result in region 2).

4 Using Non-Cooperative Rules

In this section we present several results obtained for CR P systems using non-cooperative rules; the results are obtained simulating parallel grammar devices.

If we use antiports with unbounded weight, then the simulation of ET0L systems is quite simple, as shown in the following theorem.

First we recall the basic notions about Lindenmayer systems (for a complete information we suggest [10]).

An ET0L system is a construct $G = (V, T, H, w')$, where the components fulfill the following requirements: V is the alphabet. T is the terminal alphabet. H is a finite set of of finite substitutions $H = \{h_1, h_2, \cdots, h_t\}$ (t is the number of tables); each $h_i \in H$ can be represented by a list of context-free rules $A \to x$, such that $A \in V$ and $x \in V^*$ (this list for h_i should satisfy that each symbol of V appears as the left side of some rule in h_i). $w' \in V^*$ is the axiom.

G defines a derivation relation $\Rightarrow$ by $x \Rightarrow y$ iff $y \in h_i(x)$, for some $1 \leq i \leq t$, where h_i is interpreted as a substitution mapping.

The language generated by G is $L(G) = \{w \in V^* \mid w' \Rightarrow^* w\} \cap T^*$, where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

In what follows, for simplicity, we also say that a language L is in $ET0L$ (i.e., $L \in ET0L$) if it can be generated by some ET0L system.

Moreover it is known, [10], that for each $L \in ET0L$, there exists an ET0L system G', with only 2 tables, such that $L = L(G')$.

We also need to present the following normal form for ET0L systems.

Lemma 4.1 *(Normal Form) For each $L \in ET0L$ there is an extended tabled Lindenmayer system $G = (V, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) generating L, such that the terminals are only trivially rewritten: for each $a \in T$ if $(a \to \alpha) \in h_1 \cup h_2$ then $\alpha = a$.*

A proof of this lemma can be found in [2].

In the following theorem we show how to construct a CR P system simulating an ET0L system.

Theorem 4.1 $PsET0L \subseteq PsCRP_2(0, *, ncoo)$.

Proof. Given an extended tabled Lindenmayer system $G = (V, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) in the normal form described before generating L, we construct n CR P system Π generating the Parikh set of L (actually we remove the trivial productions from h_1 and h_2). We take:

$$\Pi = (O, R, l, \mu, w_1, w_2, R_1, R_2, R_1', R_2', 2),$$

where

- $O = V \cup T \cup \{\#\}$;
- $R = h_1 \cup h_2 \cup \{\# \to \#\} \cup \{N \to \# \mid N \in V\} = R''$;
- l is a injective labeling of the rules in R;
  we use $L_1 = \{l(r) \mid r \in h_1\}$; $L_2 = \{l(r) \mid r \in h_2\}, L_3 = \{l(r) \mid r \in R''\}$;
- $\mu = [_1[_2]_2]_1$;

- $w_1 = \emptyset, w_2 = w'$;
- $R_1 = h_1 \cup R''$;
- $R_2 = h_2 \cup \{\# \to \#\}$;
- $R'_1 = \emptyset$;
- $R'_2 = \{(x, in; y, out) \mid x =< L_1 >$ and $y = \langle L_2 \rangle\} \cup \{(x, in; y, out) \mid x = \langle L_2 \rangle$ and $y = \langle L_1 \rangle\} = S_1$
  $\cup \{(y, in; x, out) \mid y = \langle L_3 \rangle$ and $x = \langle L_1 \rangle\} \cup \{(y, in; x, out) \mid y = \langle L_3 \rangle$ and $x = \langle L_2 \rangle\} = S_2$.

The system works in the following way. The rules of the two tables h_1 and h_2 are moved between regions 1 and 2 using the antiports in S_1.

In this way it is simulated the application of the productions of one of the two table over the symbol-objects contained in region 2 (at the beginning, in region 2 are present the symbol-objects corresponding to the axiom w' of G). These antiports guarantee that it is not possible to apply rules of the first table mixed with rules of the second table.

To stop the computation (in particular the movements of evolution rules across region 1 and 2) we have to use the antiports in S_2.

Using these antiports we obtain two things: the rules of R'' are moved into region 1 and on the opposite direction the rules of the table h_1 (or h_2) are moved into region 1. If in region 1 there are still symbol-objects corresponding to nonterminals of G, then one of the rules in R'' will produce the trash symbol # in region 2 and then the computation will never halt because of the presence in such region of the rule $\# \to \#$. Therefore the computation halts only when all symbol-objects corresponding to terminals of G are obtained in region 1 and then the system Π generates exactly the Parikh set of $L(G)$. □

4.1 Using Priority among Transport Rules

The previous result has been obtained using antiports of unbounded weight; in particular looking to the proof of theorem 4.1 we see that the weight of the antiport used is the maximum of the cardinality of the two tables of the ET0L system to simulate (and this number is not bounded).

Now we present a way to decrease (and to bound) the weight of the antiport rules, by using a priority among the transport rules of the system.

The priority used here is like the classical *weak priority* defined in the P system area, [8]. The difference is that here the priority is defined among transport rules in charge of moving evolution rules.

Given two sets of transport rules R_1 and R_2, to indicate that R_1 has weak priority over R_2 we write $R_1 > R_2$. This means that, in the process of assigning transport rules to objects (that in our system are evolution rules) first the transport rules in R_1 are assigned in a non-deterministic, maximally parallel manner, and then the rules in R_2 are assigned to the remaining objects (in our case evolution rules) again in a non-deterministic, maximally parallel manner.

Because we want to distinguish between this priority and classical priority among evolution rules we will use the notation pri_{tran}.

Using a weak priority among transport rules is possible to simulate an ET0L system using antiports of weight 2 and 5 membranes.

Theorem 4.2 $PsET0L \subseteq PsCRP_5(0, 2, ncoo, pri_{tran})$.

Proof. Given an extended tabled Lindenmayer system $G = (V, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) in the normal form from Lemma 4.1 generating L, we construct an CRP system Π generating the Parikh set of L (actually we remove the trivial productions from h_1 and h_2). We construct another table h_3 composed by the rules $\{N \to \# \mid N \in V\}$, where $\#$ is a new symbol. Let table h_1 have m rules, table h_2 have k rules and table h_3 have n rules. Let $max\{m, k, n\} = p$. If one of the three tables has less than p rules, then we add $D \to D$, for $D \notin V$, as many times as needed to increase the number of rules in that table to p. Then we can suppose that the cardinality of the three tables, adjusted in this way, is exactly p. Moreover we want to have "different" rules in each one of the three tables (just to have rules with different labels). Therefore we change every rule $N \to x$ in table h_1 with the rule $N \to xd_1$ where d_1 is a new symbol not in V. Also we change every rule $N \to x$ in table h_2 with the rule $N \to xd_2$ where d_2 is a new symbol not in V. In what follows, the tables h_1 and h_2 changed in the way described are called h'_1 and h'_2. For the table h_3 we do not change the rules because they are already different from the ones on the other two tables.

We construct the system

$$\Pi = (O, R, l, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, 3),$$

where

- $O = V \cup T \cup \{\#, D\} \cup \{d_1, d_2\}$;
- $R = h'_1 \cup h'_2 \cup h_3 \cup \{\# \to \#\}$;
- l is an injective labeling of the rules in R;
  we use $l_1 = l(\# \to \#)$; we also use l'_i, l''_i, l'''_i as the labels associated by l to the i-th rule in the tables h'_1, h'_2 and h_3 respectively (the cardinality of the three tables is p);
  We call $L_1 = \{l(r) \mid r \in h'_1\} = \{l'_i \mid i \in \{1, \cdots, p\}\}$; in the same way,
  $L_2 = \{l''_i \mid i \in \{1, \cdots, p\}\}, L_3 = \{l'''_i \mid i \in \{1, \cdots, p\}\}$;
- $\mu = [_1[_2[_3[_4[_5]_5]_4]_3]_2]_1$;
- $w_1 = \emptyset, w_2 = \#, w_3 = w', w_4 = \#, w_5 = \emptyset$;
- $R_1 = \{\# \to \#\}$;
- $R_2 = h'_2$;
- $R_3 = h'_1 \cup \{d_1 \to \lambda, d_2 \to \lambda, \# \to \#\}$;
- $R_4 = h_3$;
- $R_5 = \{\# \to \#\}$;
- $R'_1 = \emptyset$;
- $R'_2 = \{(l_1, in; ij, out) \mid i \in L_1, j \in L_2\}$;
- $R'_3 = \{(l'_i, in; l''_i, out) \mid i \in \{1, \cdots, p\}\} \cup \{(l''_i, in; l'_i, out) \mid i \in \{1, \cdots, p\}\}$;
- $R'_4 = \{(l'_i, in; l'''_i, out) \mid i \in \{1, \cdots, p\}\}$;

- $R'_5 = \{(ij, in; l_1, out) \mid i \in L_1, j \in L_3\}$;
- $R'_5 > R'_4; R'_2 > R'_3$.

The system Π works in the following way. Initially, in region 3, that is the output region, the symbol-objects corresponding to the axiom w' of G are present together with the rules of the table h'_1. These rules simulate the application of the rule of h_1 over the symbol-objects present in region 3 (the dummy symbols d_1's produced by such rules are immediately deleted by the rule $d_1 \to \lambda$).

At the beginning the rules of h'_2 are in region 2, while the rules of h_3 are in region 4.

When we want to pass from table 1 to table 2 (and viceversa) then we must use the antiports in R'_3. These antiports exchange each rule of one table with the corresponding rule in the other table. We must guarantee that all the rules are exchanged (i.e. the passage from one table to the other one is "complete") and that we do not have rules of the table h'_1 mixed with rules of the table h'_2 in region 3. That is checked by the antiport rules in R'_2. In fact, if the passage from one table to the other one is not correct then in region 2 rules of the table h'_1 and rules of the table h'_2 will be present at the same time. In that case one of the antiports in R'_2 is applied (it has higher priority over the antiports in R'_3) and then in region 2 the rule $\# \to \#$ is imported (because in region 2 is present the symbol $\#$ this leads to a non-halting computation). In this way we are sure that the passage from one table to the other one is made in a correct way. The presence of rules of table h'_2 in region 3 simulates the application rule of h_2 over the symbol-objects present in that region (the dummy symbols d_2's produced by such rules are immediately deleted by the rule $d_2 \to \lambda$).

The change from table h'_2 to table h'_1 is made in the way described before for the passage from table h'_1 to h'_2.

Finally, we have to guarantee that the computation halts only when all symbol-objects corresponding to terminals of G have been obtained in region 3. To halt the computation we have to stop the movements of rules that is made by the antiports in R'_3. This can be ensured by using the antiports in R'_4 that move the rules of table h'_1 into region 4 and the rules of table h_3 into region 3. These antiports exchange each rule of one table with the corresponding rule in the other table. After applying such antiports, the antiports in R'_3 cannot be applied anymore (the rules of h'_1 and h'_2 are in region 4 and 2 respectively); the "complete" passage from table h'_1 to table h_3 is guaranteed by antiports in R'_5 that work in a similar way like the one described before for antiports in R'_2 (notice that $R'_5 > R'_4$). In particular, if it happens that rules from h'_1 and from h_3 are present, at the same time, in region 4, then the rule $\# \to \#$ is moved in region 4 using one of the antiports in R'_5 and this leads to a non-halting computation (in region 4 is present the symbol-object $\#$). Therefore, also the passage from table h'_1 to h_3 must be made in a "complete" way.

On the other hand, the rules of h_3 (that are of the kind $N \to \#, N \in V$) check that in region 3 there are only terminal symbol-objects: if this is not the case, then the symbol $\#$ is produced in region 3 and, using the rule $\# \to \#$ presents in that region, a non-halting computation is obtained.

Therefore, the computation halts only when all symbol-objects corresponding to terminals of G are obtained in region 3 and then the system Π generates, in this region, exactly the Parikh set of $L(G)$. □

4.2 Simulating Indian Parallel Grammars

If we use antiports with a bounded weight (and no priorities among the transport rules) then, at least, we can generate the Parikh set of the languages generated by the *Indian parallel grammars*.

Before giving the proof of this result, we recall the definition of Indian parallel grammar (for more information about such grammars we suggest [4] and [5]).

An Indian parallel grammar is a construct $G = (N, T, S, P)$, where at each step of the derivation, every occurrence of one letter is rewritten (by using the same production).

This means that the derivations are defined in the following way: for every $x \in (N \cup T)^+$ and $y \in (N \cup T)^*$ we write $x \Rightarrow y$ if and only if $x = x_1 A x_2 A \cdots x_n A x_{n+1}, A \in N, x_i \in ((N \cup T) - A^*), 1 \leq i \leq n+1$, $y = x_1 w x_2 w \cdots x_n w x_{n+1}$, $A \to w \in P$.

We denote by IPG the family of languages generated by Indian parallel grammars.

Theorem 4.3 $PsIPG \subseteq PsCRP_2(0, 2, ncoo)$.

Proof. Given an Indian parallel grammar $G = (N, T, S, P)$ we construct a new Indian parallel grammar $G' = (N' = N \cup \{S'\}, T, S', P' = P \cup \{S' \to S\})$. Of course it is true that $L(G') = L(G)$. On the other hand, we can suppose that no trivial rules of the kind $X \to X, X \in N$, are present in P.

We construct the CR P system

$$\Pi = (O, R, l, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, 2),$$

where:

- $O = N' \cup T \cup \{Z, D\}$, with $Z, D \notin N'$;
- $R = P' \cup \{D \to D, Z \to Z\} \cup \{X \to X \mid X \in N'\}$;
- l is a injective labeling of the rules in R; we use $l_1 = l(D \to D)$, $l_2 = l(Z \to Z)$;
- $\mu = [_1[_2]_2]_1$;
- $w_1 = \emptyset, w_2 = S'$;
- $R_1 = \{S' \to S', Z \to Z\} \cup P$;
- $R_2 = \{X \to X \mid X \in N\} \cup \{S' \to S, D \to D\}$;
- $R'_1 = \emptyset$;
- $R'_2 = R''_2 \cup R'''_2 \cup R''''_2$, where
  $R''_2 = \{(x, in; y, out) \mid x = l_1 l(X \to X), y = l(X \to w) \text{ with } X \to w \in P'\}$
  $R'''_2 = \{(x, in; y, out) \mid x = l(X \to w), y = l_1 l(X \to X) \text{ with } X \to w \in P'\}$
  $R''''_2 = \{(x, in; y, out) \mid x = l_2 l(X \to X), y = l(X \to w) \text{ with } X \to w \in P'\}$.

The system Π works in the following way. In region 2 are present the symbol-objects corresponding to a sentential form generated by the grammar G'. At the beginning only the symbol-object S' is present in region 2. The idea is that, in region 2, a derivation of the grammar G' is simulated. The antiports in R'_2 are used to bring in region 2, one for time, the rules of the grammar G'. In particular, the antiports in R'''_2 are used to bring inside region 2 a new rule of G' to apply, while the antiports in R''_2 are used to

put back the rule of G' that has been used in region 2. The antiports in R_2'''' are used to stop the computation. The rules $X \to X, X \in N'$ are used to check that, when the computation halts, then no symbol-objects corresponding to nonterminals of G' are still present in region 2.

In particular, when one of the antiports in R_2''' is applied, then a rule $X \to w$ moves from region 1 to region 2 and, at the same time, the rules $D \to D$ and $X \to X$ move from region 2 to region 1. The rule $D \to D$ is used like a dummy rule: any rule of G' can move from region 1 to region 2 only using one of the antiport in R_2''' and then only when such dummy rule is present in region 2 (this dummy rule guarantees that only one rule of G' is present, at each moment, in region 2). The rule $X \to X$ is also moved from region 2 to 1 to avoid a "mixed" application, in region 2, of the rules $X \to w$ and $X \to X$.

After a rule $X \to w$ is inside region 2 then it can be applied or it can exit without being applied. Without loss of generality we suppose the rule is applied and then, in the next step, it can exit or it can be re-applied. After it has been applied a certain number of times ($0, 1$ or more) then the rule must exit and a new rule of G' must enter in region 2.

For this goal the antiports in R_2'' must be used. Using one of the antiport in R_2'' the rule $X \to w$ can move from region 2 to 1. At the same time, in the opposite direction, the dummy rule $D \to D$ and the rule $X \to X$ move back from region 1 to 2. At this point a new rule of G' can be moved inside region 2 using one of the antiport in R_2''' as described before. As already told before, only one rule of G' can be present, at each moment, in region 2, because of the use of the dummy rule $D \to D$ in the antiports R_2'''; in fact only one copy of the rule $D \to D$ is present in the system and such copy moves between region 2 and 1.

To halt the computation the movement of rules between region 2 and 1 should be stopped. This is done applying one of the antiport in R_2'''' (the last rule $X \to w$ of G' used in region 2 is moved out and, at the same time, the dummy rule $Z \to Z$ and the rule $X \to X$ are moved to region 2; notice that the dummy rule $D \to D$ is not moved back in this case). After applying such antiport the movement of rules between region 2 and 1 has been stopped (the dummy rule $D \to D$ is in region 1 and in region 2 there are no rules of G' and then no antiports in R_2''' can be applied anymore).

On the other hand, because in region 2 there are the rules $\{X \to X \mid X \in N'\}$ this guarantees that, when the computation halts, no symbol-objects corresponding to nonterminals of G' are present in region 2.

Therefore, the system Π generates in region 2 exactly the Parikh set of $L(G')$ □

5 Using Catalytic Rules: Universality

If we use catalysts, then we can inhibit the parallelism in the application of the evolution rules and we can simulate sequential grammars: in this case CR P systems become universal. In this section, we present a result where CR P systems simulate matrix grammars with appearance cheeking, using one catalyst, antiports of weight 2 and 3 membranes.

For the way CRP systems work and in particular because it is not possible to move objects, it seems not easy how to inhibit the parallelism without using catalysts.

5.1 Matrix Grammar Simulation

We recall in this section the definition and notations used for matrix grammars, with appearance checking, in the Z-binary normal form (we suppose the reader familiar with the notion of matrix grammar; for more details, the reader can consult the introduction presents in [8]). We say that a matrix grammar with appearance checking (ac) $G = (N, T, S, M, F)$ is in the Z-binary normal form if $N = N_1 \cup N_2 \cup \{S, Z, \#\}$ with these three sets mutually disjoint and the matrices in M of the following forms:

1. $m_i : (S \to X_{init} A_{init})$, with $X \in N_1, A_{init} \in N_2, i = 0$,
2. $m_i : (X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, \mid x \mid \leq 2, 1 \leq i \leq k$,
3. $m_i : (X \to Y, A \to \#)$, with $X \in N_1, Y \in N_1 \cup \{Z\}, A \in N_2, k+1 \leq i \leq n$,
4. $m_i : (Z \to \lambda), i = n+1$.

There is only one matrix of type 1, and F consists of all rules $A \to \#$ appearing in matrices of type 3; and if a sentential form in G contains Z, then it is of the form Zw, with $w \in (T \cup \{\#\})^*$.

Lemma 5.1 *For each language $L \in RE$ there is a matrix grammar with appearance checking in the Z-binary normal form such that $L = L(G)$.*

Using the previous lemma, it is possible to show the following result:

Theorem 5.1 $PsRE = PsCRP_3(0, 2, cat_1)$.

Proof. The idea is to simulate matrix grammars with ac, in the Z-binary normal form. We start from a matrix grammar with ac, in Z-binary normal form, $G = (N, T, S, M, F)$, in the standard notation. We construct the following CR P system

$$\Pi = (O, R, l, \mu, w_1, w_2, w_3, R_1, R_2, R_3, R'_1, R'_2, R'_3, 2),$$

where:

- $O = N \cup T \cup \{d, g, g', g'', g''', g^{iv}, \#\} \cup \{i \mid 1 \leq i \leq n\}$;
- $R = \{X \to iYd, cA \to icxd \mid m_i = (X \to Y, cA \to cxd) \in M, 1 \leq i \leq k\} = S_1$
  $\cup \{X \to iYd, A \to i\# \mid m_i = (X \to Y, A \to \#) \in M, k+1 \leq i \leq n\} = S_2$
  $\cup \{Z \to d \mid m_{n+1} = (Z \to \lambda)\}$
  $\cup \{g'' \to g'', g''' \to g''', g^{iv} \to g^{iv}, i \to \lambda\} = S_3$
  $\cup \{g \to g, d \to \lambda\} = S_4$
  $\cup \{g' \to g', d \to \#, \# \to \#\} = S_5$;
- l is a injective labeling of the rules in R;
  we use $r_{i,1} = l(X \to iYd)$ with $X \to iYd \in S_1 \cup S_2$,
  $r_{i,2} = l(cA \to icxd) = l(A \to i\#)$ with $cA \to icxd \in S_1, A \to i\# \in S_2$,
  $r_f = l(Z \to d)$,
  $r_0^m = l(g^m \to g^m)$ with $1 \leq m \leq 4, r_0 = l(g \to g)$.
- $\mu = [_1[_2[_3]_3]_2]_1$;

- $w_1 = \lambda$,
  $w_2 = cX_{init}A_{init}$,
  $w_3 = cZX_1X_2 \cdots X_nA_1A_2 \cdots A_m, X_i \in N_1, A_j \in N_2$;
- $R_1 = S_1 \cup S_2 \cup S_3$;
- $R_2 = S_4$;
- $R_3 = S_5$;
- $R'_1 = \emptyset$;
- $R'_2 = \{(r_{i,1}r_{i,2}, in; r_0, out) \mid 1 \leq i \leq n\} = S'_1$
  $\cup \{(r_0, in; r_{i,1}r_{i,2}, out) \mid 1 \leq i \leq n\} = S'_2$
  $\cup \{(r_f r''_0, in; r_0, out), (r'''_0, in; r''_0, out), (r^{iv}_0, in; r'''_0 r_f, out)\}$;
- $R'_3 = \{(r_f, in; r'_0, out)\}$
  $\cup \{(r_{i,j}, in; r'_0, out) \mid 1 \leq i \leq k, j \in \{1, 2\}\} = S''_1$
  $\cup \{(r_{i,1}, in; r'_0, out) \mid k + 1 \leq i \leq n\} = S''_2$.

The system Π works in the following way. Initially the evolution rules corresponding to the matrices of the matrix grammar are present in region 1 and, in particular, S_1 is the set of rules (corresponding to the matrices) of type 2, S_2 is the set of rules of type 3; moreover also the final rule $Z \to d$ is present. Each one of these rules has been modified, to generate, when applied, also a special symbol d that will be used to check if a rule has been applied. The symbol i produced by a rule when applied (except by the final rule) is only used to have all rules different (in fact, otherwise, two identical rules, in two different matrices get the same label). Such symbol i is immediately deleted when generated.

The sentential form is stored in region 2 (that is also the output region of the system); at the beginning of the computation in region 2 only the objects X_{init}, A_{init} and the catalyst c are present. The idea is that, step after step, the rewriting of the matrix grammar is simulated in region 2 over the symbol-object of the obtained sentential form. If something goes wrong during the application of the rules of the matrix grammar, then the symbol # is generated in region 3 and the computation will never halts.

We show in more details how the computation proceeds: using the antiports in S'_1 two rules of the same matrix i, with labels $r_{i,1}, r_{i,2}$, are moved from region 1 into region 2. Once the rules (with labels) $r_{i,1}, r_{i,2}$ are in region 2 then they can be applied, or they can exit together, moving from region 2 to 1 using one of the antiports in S'_2 (and in this case a new pair of rules can be introduced in region 2). If both the rules are applied, then in the next step they can be reapplied or they can exit together, as in the previous case. Because of the maximality of the parallelism and because the two rules can only exit together then, if they can be applied, they are both applied or both come back to region 1 without being applied. The presence of the "dummy" rule with label r_0 is used only to guarantee that only one pair of rule move inside region 2 (i.e., when r_0 is in region 1, then no other rules can enter in region 2).

Suppose now that only one of the two rules can be applied and let us suppose that the rules are of the matrices of type 2. Without loss of generality suppose that the first rule $r_{i,1}$ can be applied and it is applied in region 2, at step j. In the same step j, the other rule, $r_{i,2}$ cannot be applied because the corresponding nonterminal object is not present. In this case, because $r_{i,2}$ cannot come back "alone" to region 1 using one of the antiports in S'_2, and because of the maximality of the parallelism, such rule moves from region 2 to

region 3, using one of the antiports in S_1'', during step j. Once the rule $r_{i,2}$ is in region 3 then the computation will never halt because the symbol # will be generated and this guarantees the fact that both the rules of the same matrix of type 2 must be applied.

Suppose now that the two rules $r_{i,1}, r_{i,2}$ that are inside region 2 correspond to a matrix of type 3 (with ac). In this case, if the rule $r_{i,2}$ (used in the ac mode) is applied, but not the rule $r_{i,1}$, then the situation described before will happen, and the computation will never halts.

If, the rule $r_{i,1}$ is applied in step j, but not the other rule $r_{i,2}$, then in this case this rule simply "waits" that the first rule is applied. In fact, such rule cannot be moved neither to region 1 (where the two rules can move only together), nor to region 1, because there are no antiports in S_2'' that can move the rules corresponding to the rules used in ac mode. After the rule $r_{i,1}$ is applied, the two rules can move together to region 1, using antiports in S_2', or they can remain, again, in region 2. In this way the ac mode is respected.

The computation halts in the following way: it is necessary (i.e., to terminate the symport/antiport movements of rules) to bring inside region 2 the rules with label r_f and r_0'' using the antiport $(r_f r_0'', in; r_0, out)$ in R_2', at same step j. If Z is not present in region 2 then, for sure, in the step $j+1$, the antiport $(r_f, in; r_0', out)$ in R_3' is applied and this leads to a non-halting computation because the symbol # will be generated in region 3. In case that Z is present in region 2 then the rule r_f is applied in the step $j+1$, and, in the same step, the antiport $(r_0''', in; r_0'', out)$ in R_2'' is applied (then r_0''' comes inside region 2). Therefore, in the step $j+2$ the rule r_f can move from region 2 to region 1, together the rule r_0''', using the antiport $(r_0^{iv}, in; r_0''' r_f, out)$ in R_2'. Then the computation halts (no evolution rules and no symport/antiport rule can be applied in any region if and only if the derivation in G was terminal).

In this way, the system Π generates in region 2 exactly the Parikh set of $L(G)$.

□

6 Concluding Remarks and Open Problems

In this paper we have presented a new model of P systems called P system with symport/antiport of rules (in short, CR P system). The idea of this model is quite simple: during the computation, objects can evolve but they cannot be moved; at the same time, using classical symport/antiports rules, it is possible to move the evolution rules across the membranes of the system.

We have shown that, using non-cooperative rules and using antiports of unbounded weight or antiports of bounded weight and priorities among the transport rules, such systems generate at least the Parikh set of ET0L languages.

If we also use one catalyst, then CR P systems using antiports of weight 2 and 3 membranes become universal.

The paper leaves open many interesting questions. For instance, the universality has been obtained considering antiports of weight 2 (and no symports). What can we obtain considering only antiports of weight 1? Are such systems still universal? (From Example 3.1 we know that such systems can generate the Parikh set of non-semilinear languages.)

Moreover, what about CR P systems using only symport of rules? A simple example of such systems has been presented in Example 3.2. Is it possible to say more about such restricted class of systems?

On the other hand, in the case of non-cooperative evolution rules, is it possible to get the Parikh images of ET0L languages using antiports of bounded weight and no priority? We recall that, in the proof presented here, the bound on the weight of the antiports used is obtained at the price of using priorities among the communication rules.

It would be also interesting to consider two other basic variants for CR P systems: a first variant is to use multisets of rules; what happens if we do not consider rules in the set sense in the regions, but in a multiset sense? (i.e., more than one copy of a rule can be present in a region); the second variant is to use strings instead of symbol-objects.

Moreover, a more general suggestion is to use CR P P systems considering the variants already investigated for P systems with classical symport/antiports rules, [8], and for evolution-communication P systems [1, 3, 7] (for instance considering approaches where the transport rules have priority over the evolution rules – or viceversa –, or where the computation is made by applying evolution rules and transport rules in an interleaved way).

References

[1] A. Alhazov, Minimizing Evolution-Communication P Systems and ECP Automata, *New Generation Computing*, accepted.

[2] A. Alhazov, M. Cavaliere, Proton Pumping P Systems, *Membrane Computing* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933, Springer-Verlag, Berlin, 2004, 70–88.

[3] M. Cavaliere, Evolution–Communication P Systems, *Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 134–145.

[4] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[5] H. Fernau, Parallel Grammars: A Phenomenology, *Grammars*, 6, 1 (2003), 25–87.

[6] G. Ciobanu, R. Desai, A. Kumar, Membrane systems and distributed computing, *Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 187–202.

[7] S.N. Krishna, A. Păun, Some Universality Results on Evolution-Communication P Systems, *Technical Report 26, Brainstorming Week on Membrane Computing* (M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.), Research Group in Mathematical Linguistics, Rovira i Virgili University Tarragona. Available at `http://pizarro.fll.urv.es/continguts/linguistica/proyecto/reports/rep26.html`.

[8] Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002.

[9] Gh. Păun, Membrane Computing: Some Non-Standard Ideas, *Aspects of Molecular Computing, Essays Dedicated to Tom Head on the Occasion of His 70th Birthday* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950, Springer-Verlag, Berlin, 2004, 322–377.

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