Journal of Universal Computer Science, vol. 10, no. 5 (2004), 540-558
submitted: 1/4/04, accepted: 10/4/04, appeared: 28/5/04 © *J.UCS*

P Systems with Symport/Antiport of Rules

Matteo Cavaliere
(University of Sevilla, Spain)
Daniela Genova
(University of South Florida, Tampa, USA
genova@helios.acomp.usf.edu)

Abstract: Moving "instructions" instead of "data" using transport mechanisms inspired by biology is the basic idea of the computing device presented in this paper. Specifically, we propose a new class of P systems that use both evolution rules and symport/antiport rules. The idea of this kind of systems is the following: 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 a number of results using different combinations of evolution rules (catalytic, non-cooperative) and the weight of the symport/antiport rules. In particular, we show that using non-cooperative rules and antiports of unbounded weight makes it 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, these system become universal. Several open problems are also presented.

Key Words: Membrane computing, Communication, Evolution, P system, Symport, Antiport

Category: F.4.2, F.4.3

1 Introduction

We introduce a new type of P systems that is obtained by joining in an "exotic" way two well known classes of P systems: the one with evolution rules over symbol-objects and the one with symport/antiport rules (in what follows we assume the reader is familiar with the basic concepts of membrane computing, as for example presented in [Păun 2002]).

In a previous paper, [Cavaliere 2003], 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 of the application of simple evolution rules with no target indications) and communication between the regions of the system (consisting of the application of symport/antiport rules).

On the other hand, in [Păun 2004] Gh. Păun suggested a model of P systems where the rules are moved across the membranes rather than the objects processed by these rules. In the model presented in [Păun 2004] the migration of the simple evolution rules is governed by metarules existing in all regions.

Following the ideas of these two models, we propose another way to move rules across the membranes, by using the most typical transport mechanism of P systems: symport/antiport. 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 without communication targets and symport/antiport rules used to move the evolution rules across the membranes. The output of a computation is defined in the usual way: it is the number of objects produced in the output region at the end of a 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 illustrate the computation process and the proofs of results that we can consider preliminary.

In particular, we prove that when using non-cooperative evolution rules and antiports of unbounded weight, our systems can generate (at least) the Parikh images of ET0L languages. We then 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 images of the 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 images of recursively enumerable languages (specifically, we use antiports of weight two).

The paper is organized as follows. Section 2 contains a formal definition of a CR P system (P system with communication of rules). Section 3 presents two examples that illustrate the computation mechanism of a CR P system: one using antiport of rules and the other using symport of rules. All results mentioned above except universality are given in Section 4. The universality result is presented in Section 5. Finally, Section 6 contains concluding remarks and open problems.

2 Definition

Formally, we define the new variant of P systems as a system that uses evolution rules as defined in [Păun 2002] chapter 3 (but without communication targets, or equivalently, with all the communication targets fixed as "here"), and symport/antiport rules as defined in [Păun 2002] 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 noted that, unlike classical symport/antiport defined over a *multiset* of elements, here these rules are defined over a *set* of elements (labels of multiset rewriting rules).

Definition 1. A *P system with symport/antiport of rules* (*a CR P system* in short, where CR comes from "communication of rules") of degree $m \geq 1$ is a construct

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

where:

- O is the alphabet of objects;
- R is a finite set of simple evolution rules;
- l is an injective labeling of the rules in R; let $L = \{l(r) \mid r \in R\}$;
- μ is a membrane structure with m membranes (and hence m regions) injectively labeled with $1, 2, \cdots, m$;
- $w_i,\ 1 \leq i \leq m$, 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.
- $i_o \in \{1, 2, \cdots, m\}$ is the output region.

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. During the computation, symport/antiport rules constructed over the set of labels L associated with the rules in R are used to move 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. A finite number of objects and a finite number of simple evolution rules are associated with each region; a finite set of symport/antiport rules is associated with each membrane.

The rules are present in the regions 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 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 with membrane i. On the other hand, if the evolution rule is moved, then it cannot act on the objects of region i in the same step. Essentially, evolution rules and symport/antiport rules have the same "priority": they are chosen and applied in a non-deterministic maximally parallel way.

Evolution rules act on symbol-objects in the standard way; symport/antiport rules work in a standard way, as well, except 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 this 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 the following *notation*: If S is a set, then $x = \langle S \rangle$ means that x is a string representing S (i.e., x is a string obtained by concatenating elements of S in an arbitrary order, such that each element appears 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 an initially designated region (output region).

We use the 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 i, antiport rules of weight j (as usually, $m = *, i, j = *$ if the corresponding 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 Computing with CR P Systems: Two Examples

In this section we illustrate the computation process of a CR P system using two simple examples. In particular we show how to construct a CR P system generating the Parikh set of the non-semilinear language $\{a^{2^n} \mid n \geq 0\}$. In Example 3.1, antiports of weight 1 are used, while in Example 3.2 we use symports of weight 2.

3.1 Using Antiport of Rules

We construct the CR P system of degree 2

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

where:

- $O = \{A, a\}$;
- $R = \{A \to a, A \to AA\}$;
- l is an 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 numbers $\{2^n \mid n \geq 0\}$. 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. At some point of time we move the rule $A \to a$ from region 1 to region 2 and the rule $A \to AA$ from region 2 to region 1 using the antiport $(l_1, in; l_2, out)$ associated with membrane 2. Once the rule $A \to a$ is inside region 2 all copies of A are transformed into copies of a and no rule can be applied anymore in any region. Then the computation halts and we obtain the result in region 2.

3.2 Using Symport of Rules

Let us consider the CR P system of degree 2

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

where:

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

At the beginning of the computation only one copy of the symbol-object A is present in region 2. The rules $A \to AA$ and $C \to C$ are, also, present in region 2. In this region, the rule $A \to AA$ is applied an arbitrary number of times (notice that the rule $C \to C$ cannot be applied). At some point of 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 this step all evolution rules (namely $A \to AA, C \to C$, and $A \to a$) are in region 1. Since the symport rules are applied in a maximally parallel manner, 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. Note that, because of the dummy rule $C \to C$, this symport rule can be applied only after the rule $A \to AA$ exits from region 2. When the rule $A \to a$ enters region 2 all symbol-objects A are changed to a and the computation halts. The result, equal to the Parikh set of the language $\{a^{2^n} \mid n \geq 0\}$, is obtained in region 2.

4 Using Non-Cooperative Evolution Rules

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

First we recall the basic notions about Lindenmayer systems (for a comprehensive presentation we suggest [Rozenberg and Salomaa 1997]).

An ET0L system is a construct $G = (N, T, H, w')$, where the components fulfill the following requirements: N is the nonterminal alphabet, T is the terminal alphabet (the two alphabets are disjoint and let $V = N \cup T$), H is a finite set of finite substitutions (tables) $H = \{h_1, h_2, \cdots, h_t\}$ (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 the requirement that each symbol of V appears as the left side of some rule in h_i) – and $w' \in V^*$ 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 $\Rightarrow$.

We denote by $ET0L$ the family of languages generated by ET0L systems. It is known (see [Rozenberg and Salomaa 1997]) that for each $L \in ET0L$ there exist an ET0L system G with only 2 tables, such that $L = L(G)$.

We will use below the following normal form.

Lemma 2. *For each $L \in ET0L$ there is an extended tabled Lindenmayer system $G = (N, T, H, w')$ 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 [Alhazov and Cavaliere 2004].

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

Theorem 3. *$PsET0L \subseteq PsCRP_2(0, *, ncoo)$.*

Proof. Given an extended tabled Lindenmayer system $G = (N, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) in the normal form from Lemma 2, we construct a CR P system Π generating the Parikh set of $L(G)$ as follows. Consider

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

where:

- $O = V \cup \{\#\}$;
- $R = h_1 \cup h_2 \cup \{\# \to \#\} \cup R''$; where $R'' = \{Y \to \# \mid Y \in N\}$;
- l is an injective labeling of the rules in R;
  let $L_1 = \{l(r) \mid r \in h_1\}$; $L_2 = \{l(r) \mid r \in h_2\}$, and $L_3 = \{l(r) \mid r \in R''\}$;

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

The system works as follows. The rules of the two tables h_1 and h_2 are moved between regions 1 and 2 using the antiports in S_1. This simulates the application of the productions of one of the two table over the symbol-objects contained in region 2 (in the beginning the symbol-objects corresponding to the axiom w' of G are present in region 2). These antiports guarantee that it is not possible to apply rules of the first table mixed with rules of the second table.

The role of the antiports in S_2 is to stop the computation (in particular, the movements of evolution rules across membrane 2). The purpose of these antiports is twofold: the rules of R'' are moved into region 2 and in the opposite direction the rules of the table h_1 (or h_2) are moved into region 1. If there are still symbol-objects corresponding to nonterminals of G in region 2, then one of the rules in R'' will produce the trash symbol # and then the computation will never halt, because of the presence of the rule $\# \to \#$ in region 2. Therefore, the computation halts if and only if all symbol-objects corresponding to terminals of G are obtained in region 2 and then the system Π generates exactly the Parikh set of $L(G)$. $\square$

4.1 Using Priority among Transport Rules

The previous result was obtained using antiports of unbounded weight; in particular, the proof of Theorem 3 indicates that the weight of the antiport used is the maximum of the cardinality of the two tables of the ET0L system we have to simulate (and this number can be arbitrarily large).

We present now a way to decrease (and to bound) the weight of the antiport rules by using a priority relation among the transport rules of the system.

The priority used here is similar to the classical *weak priority* defined in the P system area (see [Păun 2002]). 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, we indicate that R_1 has weak priority over R_2 by writing $R_1 > R_2$. This means that in the process of assigning transport rules to "objects" (which 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.

In order to distinguish between this priority and the classical priority among evolution rules we use the notation pri_{tran}.

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

Theorem 4. $PsET0L \subseteq PsCRP_5(0, 2, ncoo, pri_{tran})$.

Proof. Given an extended tabled Lindenmayer system $G = (N, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) in the normal form from Lemma 2, we construct a CR P system Π generating the Parikh set of $L(G)$. After removing the trivial productions from h_1 and h_2, construct another table h_3 composed by the rules $\{Y \rightarrow \# \mid Y \in N\}$, where $\#$ is a new symbol not in V. Let table h_1 have m rules, table h_2 have k rules, and table h_3 have n rules. Let $max\{m, k, n\} = p$. Then, if one of the three tables has less than p rules, we add $D \rightarrow D$, for $D \notin V$, as many times as needed to increase the number of rules in that table to p. Then we can assume that the cardinality of the three tables so adjusted is exactly p. Moreover, we need different rules in each of the three tables (more precisely, we need rules with different labels). Therefore, we substitute every rule $X \rightarrow x$ in table h_1 with the rule $X \rightarrow xd_1$, where d_1 is a new symbol not in V. Similarly, we substitute every rule $X \rightarrow x$ in table h_2 with the rule $X \rightarrow xd_2$ where d_2 is a new symbol not in V. In what follows, the tables h_1 and h_2 changed in the above described way 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 in the other two tables.

Consider the CR P system

$$\Pi = (O, R, l, \mu, w_1, w_2, w_3, w_4, w_5, R_1, R_2, R_3, R_4, R_5, R'_1, R'_2, R'_3, R'_4, R'_5, 3),$$

where:

- $O = V \cup \{\#, D\} \cup \{d_1, d_2\}$;
- $R = h'_1 \cup h'_2 \cup h_3 \cup \{\# \rightarrow \#\}$;
- l is an injective labeling of the rules in R;
  let $l_1 = l(\# \rightarrow \#)$, let l'_i, l''_i, l'''_i be the labels associated with the i-th rule in the tables h'_1, h'_2 and h_3, respectively (the cardinality of the three tables is p), and let $L_1 = \{l(r) \mid r \in h'_1\} = \{l'_i \mid i \in \{1, \cdots, p\}\}$; similarly,
  $L_2 = \{l''_i \mid i \in \{1, \cdots, p\}\}$, and $L_3 = \{l'''_i \mid i \in \{1, \cdots, p\}\}$;

- $\mu = [_1[_2[_3[_4[_5\ ]_5]_4]_3]_2]_1$;
- $w_1 = \emptyset, w_2 = \#, w_3 = w', w_4 = \#, w_5 = \emptyset$;
- $R_1 = \{\# \rightarrow \#\}$;
- $R_2 = h'_2$;
- $R_3 = h'_1 \cup \{d_1 \rightarrow \lambda, d_2 \rightarrow \lambda, \# \rightarrow \#\}$;
- $R_4 = h_3$;
- $R_5 = \{\# \rightarrow \#\}$;
- $R'_1 = \emptyset$;
- $R'_2 = \{(l_1, in; ij, out) \mid i \in L_1, j \in L_2\}$;
- $R'_3 = \{(l'_i, in; l''_i, out) \mid i \in \{1, \cdots, p\}\} \cup \{(l''_i, in; l'_i, out) \mid i \in \{1, \cdots, p\}\}$;
- $R'_4 = \{(l'_i, in; l'''_i, out) \mid i \in \{1, \cdots, p\}\}$;
- $R'_5 = \{(ij, in; l_1, out) \mid i \in L_1, j \in L_3\}$;
- $R'_5 > R'_4; R'_2 > R'_3$.

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

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

In order to pass from table 1 to table 2 (and viceversa) we use the antiports in R'_3. These antiports exchange each rule of one table with a rule in the other table. We need to guarantee a full "swap" of the rules (i.e., that all rules from one table are exchanged with all rules of the other table). Ensuring that the passage from one table to the other is "complete" is essential, since we need to avoid having rules of table h'_1 mixed with rules of 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 rules from both tables will be present in region 2 simultaneously. In that case, one of the antiports in R'_2 is applied (it has a higher priority over the antiports in R'_3) and then the rule $\# \rightarrow \#$ is imported in region 2 (because the symbol $\#$ is present in region 2, this leads to a non halting computation). Thus, such a construction ensures that the passage from one table to the other one is made in a correct way.

The rules of table h'_2 from region 3 simulates the application of a rule from 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 $d_2 \rightarrow \lambda$). The change from table h'_2 to table h'_1 is made similarly to the above described way for the passage from table h'_1 to h'_2.

Finally, we need to guarantee that the computation halts only when all symbol-objects in region 3 have been transformed into terminals of G. To halt the computation we have to stop the movements of rules made possible by the antiports in R'_3. This can be made 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 similarly to the ones described above 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 simultaneously present in region 4, then the rule $\# \rightarrow \#$ is moved to region 4 using one of the antiports in R'_5 and this leads to a non halting computation. Therefore, the passage from table h'_1 to h_3 must be made in a "complete" way, as well.

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

Therefore, the computation halts iff all symbol-objects corresponding to terminals of G are obtained in region 3. Thus, the system Π generates (in region 3) exactly the Parikh set of $L(G)$. $\square$

4.2 Simulating Indian Parallel Grammars

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

Before presenting the proof of this result, we recall the definition of an Indian parallel grammar (for a more detailed discussion of such grammars we refer the reader to [Dassow and Păun 1989] and [Fernau 2003]).

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 using the same production.

This means that the derivations are defined in the following way: for every $x, y \in (N \cup T)^+$ we write $x \Rightarrow y$ if and only if $x = x_1Ax_2A \cdots x_nAx_{n+1}$, for some $A \in N, x_i \in ((N \cup T) - A^*), 1 \leq i \leq n+1$, and $y = x_1wx_2w \cdots x_nwx_{n+1}$, for $A \rightarrow w \in P$.

The language generated by G is $L(G) = \{w \in (N \cup T)^* \mid S \Rightarrow^* w\} \cap T^*$, where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

The family of languages generated by Indian parallel grammars is denoted by IPG.

Theorem 5. $PsIPG \subseteq PsCRP_2(0, 2, ncoo)$.

Proof. Given an Indian parallel grammar $G = (N, T, S, P)$, we construct a new Indian parallel grammar $G' = (N' = N \cup \{S'\}, T, S', P' = P \cup \{S' \to S\})$. It is obvious that $L(G') = L(G)$. In addition, we assume that P does not contain trivial rules of the kind $X \to X, X \in N$.

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

The system Π works in the following way. The symbol-objects corresponding to a sentential form generated by the grammar G' are present in region 2. At the beginning, only the symbol-object S' is present in region 2. The basic idea is to simulate a derivation of the grammar G' in region 2. The antiports in R_2' are used to bring in region 2, one at a 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 be applied, while the antiports in R_2'' are used to bring back the rule of G' that has been used in region 2. The antiports in R_2^{iv} are used to stop the computation. The rules $X \to X, X \in N'$, are used to check that no symbol-objects corresponding to nonterminals of G' are present in region 2 at the time the computation halts.

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 in the opposite direction. A rule from P' can move from region 1 to region 2 only by using one of the antiports in R_2''' and then only when the dummy rule $D \to D$ is present in region 2. The role of the dummy rule $D \to D$ is to guarantee that only one rule of P' is present in region 2 at any particular moment of time. The rule $X \to X$ is also moved from region 2 to 1 to avoid a "mixed" application of the rules $X \to w$ and $X \to X$ in region 2.

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

To achieve this, the antiports in R_2'' must be used. Using one of the antiports 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 from P' can be moved inside region 2 using one of the antiports in R_2''' as described above. As mentioned earlier, only one rule of G' can be present in region 2 at each moment, because of the dummy rule $D \to D$ in the antiports of R_2'''. In fact, only one copy of the rule $D \to D$ is present in the system and this copy moves between regions 2 and 1.

To halt the computation the movement of rules between regions 2 and 1 should be stopped. This can be achieved by applying one of the antiports in R_2^{iv}. The last rule $X \to w$ from P' 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 an antiport, the movement of rules between regions 2 and 1 stops. At that time, the dummy rule $D \to D$ is in region 1 and there are no rules from P' in region 2. Thus, no antiports from R_2''' can be applied anymore.

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

Therefore, the system Π generates exactly the Parikh set of $L(G')$. $\square$

5 Using One Catalyst: 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 computationally universal. In this section we present an universality result where CR P systems simulate matrix grammars with appearance checking using one catalyst, antiports of weight 2, and 3 membranes.

Because of the way CR P systems work and in particular since it is not possible to move objects, inhibiting the parallelism without using catalysts seems a very hard task.

5.1 Matrix Grammar Simulation

In this section we recall the definition and notations used for matrix grammars with appearance checking in the Z-binary normal form (we assume that the reader is familiar with the notion of a matrix grammar; for more details, the reader can consult the introduction presented in [Păun 2002]). 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 $X_{init} \in N_1, A_{init} \in N_2, i = 0$,
2. $m_i : (X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2, 1 \leq i \leq k$,
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, F consists of all rules $A \to \#$ appearing in the matrices of type 3, and if a sentential form in G contains Z, it is of the form Zw, with $w \in (T \cup \{\#\})^*$.

Lemma 6. *For each language $L \in RE$ there is a matrix grammar with appearance checking in the Z-binary normal form such that $L = L(G)$.*

We use the above lemma to show the following result:

Theorem 7. $PsRE = PsCRP_3(0, 2, cat_1)$.

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

- $O = N \cup T \cup \{d, g, g', g'', g''', g^{iv}, \#\} \cup \{i \mid 1 \leq i \leq n\}$;

– $R = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5$, where
$S_1 = \{X \to iYd, cA \to icxd \mid m_i = (X \to Y, cA \to cxd) \in M, 1 \leq i \leq k\}$,
$S_2 = \{X \to iYd, A \to i\# \mid m_i : (X \to Y, A \to \#) \in M, k+1 \leq i \leq n\}$,
$S_3 = \{Z \to d \mid m_{n+1} : (Z \to \lambda)\} \cup \{g'' \to g'', g''' \to g''', g^{iv} \to g^{iv}, i \to \lambda\}$,
$S_4 = \{g \to g, d \to \lambda\}$,
$S_5 = \{g' \to g', d \to \#, \# \to \#\}$;

– l is an injective labeling of the rules in R;
let $r_{i,1} = l(X \to iYd)$ 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 $1 \leq m \leq 4, r_0 = l(g \to g)$;

– $\mu = [_1[_2[_3\ ]_3]_2]_1$;

– $w_1 = \lambda$,
$w_2 = cX_{init}A_{init}$,
$w_3 = cZX_1X_2 \cdots X_{m'}A_1A_2 \cdots A_{m''}, X_i \in N_1, A_j \in N_2$;

– $R_1 = S_1 \cup S_2 \cup S_3$;

– $R_2 = S_4$;

– $R_3 = S_5$;

– $R'_1 = \emptyset$;

– $R'_2 = S'_1 \cup S'_2 \cup \{(r_f r''_0, in; r_0, out), (r'''_0, in; r''_0, out), (r^{iv}_0, in; r'''_0 r_f, out)\}$, where
$S'_1 = \{(r_{i,1}r_{i,2}, in; r_0, out) \mid 1 \leq i \leq n\}$,
$S'_2 = \{(r_0, in; r_{i,1}r_{i,2}, out) \mid 1 \leq i \leq n\}$;

– $R'_3 = S''_1 \cup S''_2 \cup \{(r_f, in; r'_0, out)\}$, where
$S''_1 = \{(r_{i,j}, in; r'_0, out) \mid 1 \leq i \leq k, j \in \{1,2\}\}$,
$S''_2 = \{(r_{i,1}, in; r'_0, out) \mid k+1 \leq i \leq n\}$.

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. In particular, S_1 is the set of rules corresponding to the matrices of type 2, S_2 is the set of rules corresponding to matrices of type 3. Moreover, the final rule $Z \to d$ is also present. Each one of these rules has been modified to generate a special symbol d that will be used to check if a rule has been applied. The purpose of the symbol i produced by a rule when applied (except by the final rule) is to distinguish between the rules (otherwise, two identical rules in two different matrices get the same label). This symbol i is immediately deleted when generated.

The sentential form is stored in region 2, the output region of the system. At the beginning of the computation only the objects X_{init}, A_{init} and the catalyst

c are present in region 2. The idea is that, step after step, the productions of the matrix grammar are applied to the symbol-objects present in region 2. 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 halt.

We now show in a greater detail 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, they can be applied or they can exit together moving from region 2 to region 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 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, if they can be applied, both are applied or both come back to region 1 without being applied. The "dummy" rule with label r_0 is used only to ensure that only one pair of rules moves inside region 2 (i.e., when r_0 is in region 1 then no other rules can enter region 2).

Now suppose that only one of the two rules can be applied and let us assume that the rules are associated with a matrix 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, during step j, this rule moves from region 2 to region 3, using one of the antiports of S_1''. Once the rule $r_{i,2}$ is in region 3, the computation will never halt, because the symbol # will be generated. In this way we make sure that both rules of the matrix of type 2 are applied.

Suppose now that the two rules $r_{i,1}$ and $r_{i,2}$ 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 above will occur and the computation will never halt. If the rule $r_{i,1}$ is applied in step j but not the other one $r_{i,2}$, then this rule simply "waits" that the first rule is applied. In fact, this rule cannot be moved neither to region 1 where the two rules can move only together, nor to region 3, 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 in region 2 again. In this way the ac mode is respected.

In order to halt the computation (i.e., to terminate the symport/antiport movements of rules) it is necessary to bring the rules with labels r_f and r_0'' inside region 2 using the antiport $(r_f r_0'', in; r_0, out)$ in R_2' at some step j. If Z is not present in region 2, then in 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 the case that Z is present in region 2, 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 with the rule r_0''' using the antiport $(r_0^{iv}, in; r_0''' r_f, out)$ in R_2'. If the current multiset contains only terminal objects (that is, # is not present), then the computation halts (no evolution rules and no symport/antiport rule can be applied in any region).

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

6 Concluding Remarks and Open Problems

In this paper we have presented a new type of P systems called P systems with symport/antiport of rules (CR P systems in short). 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, evolution rules are moved across the membranes of the system.

We have shown that, using non-cooperative rules and 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 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 some non-semilinear languages.)

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

On the other hand, is it possible to get the Parikh images of ET0L languages in the case of non-cooperative evolution rules, using antiports of a 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.

Also, it would be interesting to consider two other basic variants of CR P systems: the 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.

A more general suggestion is to use CR P systems considering the variants already investigated for P systems with classical symport/antiport rules

[Păun 2002] and for evolution-communication P systems [Alhazov 2003], [Cavaliere 2003, Krishna and Păun 2003] (for instance, considering approaches where the symport/antiport rules have priority over the evolution rules (or viceversa)).

A possible variant of CR P systems, studied in [Freund and Oswald 2004], is obtained by dividing the computation into two substeps that are applied in an interleaved way: first all possible communication rules are applied, and then the evolution rules are applied, in a non-deterministic, maximally parallel manner. In this case, using non-cooperative evolution rules, the systems generate exactly the Parikh set of ET0L languages. Is it possible to prove the same upper bound for CR P systems? Actually, using our definition of CR P systems, this seems harder because of the mixed application of communication and evolution rules. On the other hand, in [Freund and Oswald 2004] the universality is obtained using only one pure catalyst. Is it possible to obtain the same result in the case of CR P systems? These are some of the questions that remain to be answered in the future.

We would like to conclude this paper with some (maybe) "philosophical" considerations: moving rules instead of objects is, somehow, a new approach for 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 in several papers; see [Ciobanu et al. 2003]). It would be interesting to compare such a "non-standard" approach with the classical ones, investigating possible advantages and disadvantages.

References

[Alhazov 2003] Alhazov A.: "Minimizing Evolution-Communication P Systems and ECP Automata"; New Generation Computing (2003), accepted

[Alhazov and Cavaliere 2004] Alhazov A., Cavaliere M.: "Proton Pumping P Systems"; "Membrane Computing. International Workshop WMC2003. Revised Papers" (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Lecture Notes in Computer Science 2933, Springer, Berlin (2004), 70–88

[Cavaliere 2003] Cavaliere M.: "Evolution–Communication P Systems"; "Membrane Computing. International Workshop WMC2002. Revised Papers" (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Lecture Notes in Computer Science 2597, Springer, Berlin (2003), 134–145

[Ciobanu et al. 2003] Ciobanu G., Desai R., Kumar A.: "Membrane systems and distributed computing"; "Membrane Computing. International Workshop WMC2002. Revised Papers" (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Lecture Notes in Computer Science 2597, Springer, Berlin (2003), 187–202

[Dassow and Păun 1989] Dassow J., Păun Gh.: "Regulated Rewriting in Formal Language Theory"; Springer, Berlin (1989)

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

[Freund and Oswald 2004] Freund R., Oswald M.: "P Systems with Antiport Rules for Evolution Rules"; Technical Report 01/2004, "Brainstorming Week on Membrane Computing 2004", Research Group on Natural Computing, University of Sevilla, Sevilla (2004).

[Krishna and Păun 2003] S.N. Krishna, A. Păun: "Some Universality Results on Evolution-Communication P Systems"; Technical Report 26, Brainstorming Week on Membrane Computing 2003 (M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.), Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona (2003)

[Păun 2002] Păun Gh.: "Membrane Computing. An Introduction"; Springer, Berlin (2002)

[Păun 2004] Păun Gh.: "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.), Lecture Notes in Computer Science 2950, Springer, Berlin (2004), 322–377

[Rozenberg and Salomaa 1997] Rozenberg G., Salomaa A. (eds.): "Handbook of Formal Languages"; Springer, Berlin (1997)