Evolution–Communication P Systems

Matteo Cavaliere*

Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tarraco 1, 43005 Tarragona, Spain
mc1.doc@estudiants.urv.es

Abstract. We propose a new class of P systems that use simple evolution rules (classical evolution rules without communication targets) and symport/antiport rules (for communication).
This type of P system is realistic for (at least) three different reasons: we do not have target indications in the evolution rules, we use very simple symport/antiport rules to realize communication, and we do not need objects available in the environment at the beginning of a computation. Somewhat expected, this new variant is still universal. We prove the universality in two cases: when using catalytic rules (but only one catalyst), symport/antiport rules of weight one, and two membranes, and when we use three membranes, symport/antiport rules of weight one, and no catalyst. Especially the latter result is of interest, because the catalysts were used in most universality proofs for P systems with symbol-objects. Also new is the proof technique we use: we start from programmed grammars with unconditional transfer.

1 Introduction

We introduce and investigate here a new class of P systems that joins two basic variants of P systems: the one with evolution rules, the other one with symport/antiport rules (in what follows, we assume the reader familiar with the basic elements of membrane computing, for instance, as presented in [5]). Moreover, in this variant, the evolution rules lose their target indications (in this way, the model becomes more realistic from a biological point of view) and the system is embedded in an empty enviroment (and not infinite as in the case of systems with symport/antiport rules). The idea of this new variant is to "split" the work of a P system in two phases: the *evolution* of the symbol-objects (application of evolution rules) and the *communication* between the regions of the system (application of the symport/antiport rules). In such a new model, we can have many different strategies to define a computation; in this paper we study the evolution-communication P systems where the computation takes place with a *mixed* "sequence" of evolution rules and symport/antiport rules. We show that

* This work was possible thanks to a research grant Beca URV from Rovira i Virgili University.

such a variant, using non-cooperative rules, low passage of information (symport/antiport rules of weight equal to one) and two membranes can generate more than the Parikh image of context-free languages. Moreover, we show that, if we use one catalyst, low symport/antiport rules (symport/antiport rules of weight equal to one) and two membranes, then we can generate all the recursively enumerable sets of integer numbers. This result is interesting in view of the fact that in [6] it has been shown that a catalytic system, using six catalyst objects and two membranes, is universal. Here, we decrease the number of catalyst objects to one, at the price of using communication by mean of simple symport/antiport rules (of weight one). Also, we show that, if we use three membranes, then we can generate all the recursively enumerable sets of integer numbers using simple symport/antiport rules of weight one and simple non-cooperative evolution rules – hence no catalyst. The proof is based on simulating programmed grammars with unconditional transfer by means of P systems. As far as we know, it is the first time when the programmed grammars are used in membrane computing proofs. Up to now, the "standard" tools for obtaining universality results were matrix grammars with appearance checking and register machines.

We recall that another approach to avoid the use of target indications in the evolution rules, and having communication mixed with evolution, has been studied in [1]; However, while in [1] one changes the *structure* of the evolution rules, here we put together two already well-studied variants of P systems.

2 EC P Systems

We define the new variant of P systems, as asystems that use evolution rules as defined in [5], chapter 3 (but without communication targets, or, equivalently, with all the communication targets fixed as "here") and symport/antiport rules, as defined in [5], chapter 4. For simplicity, we often call the evolution rules without communication targets *simple evolution rules* (or *simple catalytic rules*).

Definition 1. *An* evolution-communication P system *(in short,* an EC P system*), of degree $m \geq 1$, is defined as*

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

where:

- *O is the alphabet of objects;*
- *μ is a membrane structure with m membranes (and hence m regions) injectively labelled 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, $1 \leq i \leq m$, 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 $u \rightarrow v$, where u and v are strings over the alphabet O;*

– *R'_i, $1 \leq i \leq m$, are finite sets of symport/antiport rules over O; R'_i is associated with the membrane i of μ;*
– *$i_o \in \{0, 1, 2, \cdots, m\}$ 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.*

The m-tuple of multisets of objects present at any moment in the regions of Π represents the configuration of the system at that moment (the m-tuple $(w_1, \cdots, w_m)$ is the initial configuration). A transition between configurations is governed by the mixed application of the evolution rules and of the symport/antiport rules. All objects which can be the "subject" of the rules from the sets R_i, R'_j, $1 \leq i \leq m, 1 \leq j \leq m$, have to evolve by such rules. As usual, the rules from R_i are applied to objects in region i and the rules from R'_i govern the communication of objects through membrane i. There is no difference between evolution rules and communication rules: they are chosen and applied in the non-deterministic maximally parallel manner. The system continues parallel steps until there remain no applicable rules (evolution rules or symport/antiport rules) in any region of Π. Then the system halts, and we consider the number of objects contained in the output region i_o, at the moment when the system halts, as the result of the computation of Π. This way to have a computation in an EC P system will be called the *mixed approach.*

We use the notation

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

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

3 Computational Power: Preliminary Results

Before presenting the universality of EC P systems, we discuss some results which either follow directly from the definitions, or shed some light about the power of systems with a small number of membranes. We are interested in computational results about EC P systems that use symport/antiport rules of a low weight (systems with "long enough" symport/antiport rules are universal [5]) and with not "too powerful" evolution rules (we know that with cooperative rules or with catalyst rules, with "enough" catalyst objects, we have already universality [5], [6]). So, we will try to stay in the middle, using a combination of not so powerful symport/antiport rules and simple evolution rules. We recall that $NOP_m(\alpha, tar), PsOP_m(\alpha, tar), \alpha \in \{coo, ncoo\} \cup \{cat_k \mid k \geq 0\}$, denote the family of sets of natural numbers and the family of sets of vectors of natual numbers generated by symbol-object P systems of degree at most m, using rules of the type α, and target communications of the type $\{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

As above, we use cat_k to indicate that the corresponding P system uses at most k catalysts.

Because communication targets can, obviously, simulate the control of communication as done by a symport rule of weight 1, we have:

$$NECP_m(1, 0, \alpha) \subseteq NOP_m(\alpha, tar),$$

for all $\alpha \in \{coo, ncoo\} \cup \{cat_k \mid k \geq 0\}$.

From [6] we know that $NOP_2(cat_6, tar) = NRE$, and from [5] we have $NOP_m(ncoo, tar) = NCF$, for all $m \geq 1$. (As usual, if FL is a family of languages, then we denote by $PsFL$ the family of Parikh images of languages in FL and by NFL the family of length sets of languages in FL.)

We know that, in the case of $NOP_1(ncoo, tar)$, the role of the target indications is only to send out (this means to "lose") some symbol-objects, using target indications of the form a_{out}, where a is a symbol-object. We can simulate the evolution rules which contain such a target indication by replacing each a_{out} with λ; in this way, we can avoid to use target indications. Thus, we have

$$NOP_1(ncoo, tar) \subseteq NECP_1(0, 0, ncoo) \subseteq NECP_1(1, 0, ncoo).$$

Therefore $NECP_1(1, 0, ncoo) = NCF$.

Such an equality is no longer valid if we consider the Parikh sets and we use more powerful ingredients: symport/antiport rules of weight one and two membranes.

In the next theorem we show that, using two membranes, non-cooperative rules, and symport/antiport rules of weight one, we can generate more than the Parikh images of the context-free languages.

The idea of the proof is to simulate a programmed grammar without appearance checking (we know [2] that such grammars generate non semilinear languages, hence languages whose Parikh image is not in $PsCF$). The family of languages generated by programmed grammars with λ rules and without appearance checking is denoted by PR.

Theorem 1. $PsPR \subseteq PsECP_2(1, 1, ncoo)$.

Proof. Consider the programmed grammar $G = (N, T, P, S)$ without appearance checking. We denote by $l(S)$ the set of labels of rules of the form $(k : S \rightarrow x, \sigma(k)) \in P$. We add to P the triple $(0 : U \rightarrow S, \sigma(0))$ with $\sigma(0) = l(S)$, where U is a new non-terminal. We denote by G' the obtained grammar (N', T, P', U), where $N' = N \cup \{U\}$. Clearly, $L(G) = L(G')$ and each derivation in G' starts with the rule with label 0.

For each $x \in N'$ we consider a new symbol $\bar{x}$; denote by $\bar{N}$ the set of such symbols, and define the morphism $\gamma : (N' \cup T)^* \longrightarrow (\bar{N} \cup T)^*$ in the following way: $\gamma(x) = \bar{x}$, if $x \in N'$, and $\gamma(x) = x$, if $x \in T$.

We construct the EC P system

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

with:

$$\begin{aligned}
O &= N' \cup T \cup \bar{N} \cup \{d_i, d'_i \mid i \in lab(P')\} \cup \{\#, f, c, c_1\},\\
\mu &= [_1[_2\ ]_2]_1,\\
w_1 &= fU,\\
w_2 &= c,\\
i_o &= 0,\\
R_2 &= \{X \to \gamma(x)d_k \mid (k : X \to x, \sigma(k)) \in P'\} \cup \{d_i \to \# \mid i \in lab(P')\}\\
&\quad \cup \{\# \to \#, c_1 \to c\} \cup \{Y \to \bar{Y} \mid Y \in N'\},\\
R_1 &= \{d_i \to d'_i \mid i \in lab(P')\} \cup \{c \to c_1\} \cup \{\bar{Y} \to Y \mid Y \in N'\},\\
R'_2 &= \{(d_i, out; d'_j, in) \mid i \in \sigma(j)\} \cup \{(d_0, out; f, in), (c_1, in)\}\\
&\quad \cup \{(a, out) \mid a \in T\} \cup \{(c, out; Y, in), (\bar{Y}, out) \mid Y \in N'\},\\
R'_1 &= \{(a, out) \mid a \in T\}.
\end{aligned}$$

The EC P system Π simulates the programmed grammar G' in the following way. The evolution rules simulate the context-free rules of G', while the symport/antiport rules check if the evolution rules are applied according to the order defined by the labels in the success fields of G'. The symbol-object c, together with (one of) the antiport rules $(c, out; Z, in)$, for some $Z \in N'$, associated with membrane 2 are used; this makes sure that in each step only one object $Z \in N'$ is present in region 2, and, therefore, only one simple evolution rule of the form

$$Z \to \gamma(z)d_m, \text{ for } (m : Z \to z, \sigma(m)) \in P',$$

can be applied.

Assume that such a rule is applied in region 2 and the object d_m is produced. This object indicates that the rule of G' with label m has been applied.

After the object d_m has been produced, it must exit from region 2 (because of the evolution rule $d_m \to \#$ that produces the trap-symbol $\#$). But the object d_m can exit only if the “right” predecessor is present in region 1 (“right” in the sense of the order defined by the success fields of G'). In fact, to exit from region 2, the object d_m must use (one of) the antiport rules associated with membrane 2, that is, $(d_m, out; d'_j, in)$, for j such that $m \in \sigma(j)$. This antiport rule checks if the simple evolution rule has been applied in the correct order, as defined by the programmed grammar (the object f is used only to start the computation).

When the object d_m goes to region 1, it is changed in d'_m and this object will be the new predecessor (this means that it “stores” the label of the last applied rule) and the computation will be continued. The evolution rules $Y \to \bar{Y}$, for $Y \in N'$, present in region 2, makes sure that when the computation halts, there are no objects of N' present in regions 1 or 2.

In each step, each produced terminal is sent out by the symport rules associated with membrane 1 and 2, and we collect the output in the enviroment. Thus, modulo the order of symbols, the system Π sends out exactly the strings of $L(G) = L(G')$. □

Corollary 1. $PsCF \subset PsECP_2(1, 1, ncoo)$.

Proof. Because [2] $PsCF \subset PsPR$, the theorem implies the corollary. □

4 Universality of EC P Systems: A First Result

In this section we give a first universality theorem for EC P systems. This result is not a surprise, because the model joins two different models that are known to be universal. The interesting fact is that we can prove that our systems are universal even when they use very small symport/antiport rules, only one catalyst, and two membranes. The proof is based on the simulation of programmed grammars with appearance checking and with unconditional transfer. Such a grammar is an usual programmed grammar $G = (N, T, P, S)$, with the rules of P of the form $(i : A \to x, \sigma(i), \varphi(i))$ with $\sigma(i) = \varphi(i)$ (the success and the failure fields coincide). We recall from [3] that such grammars are able to generate the recursively enumerable one-letter languages.

Theorem 2. $NECP_2(1, 1, cat_1) = NRE$.

Proof. Let us consider a programmed grammar $G = (N, \{a\}, P, S)$ with appearance checking and with unconditional transfer. We add to P the production $(0 : U \to S, \sigma(0), \varphi(0))$ with $\sigma(0) = \varphi(0) = l(S)$, where U is a new non-terminal and $l(S)$ the set of labels of S-productions from P. In this way, we obtain a new grammar, $G' = (N', \{a\}, P', U)$, with $N' = N \cup \{U\}$. Clearly, $L(G) = L(G')$ and each derivation in G' starts with the rule with label 0.

We construct the EC P system

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

with

$$\begin{aligned}
O &= N' \cup \{d_i, d'_i, d''_i, e_i \mid i \in lab(P')\} \\
&\quad \cup \{a, F, F', F'', c, f, g, g', \#, h, p_1, p_2, \cdots, p_5\}, \\
\mu &= [_1 [_2\ ]_2]_1, \\
w_1 &= f, \\
w_2 &= cUhF, \\
i_o &= 0, \\
R_1 &= \{d_i \to d'_i, d''_i \to d'_i e_i \mid i \in lab(P')\} \\
&\quad \cup \{Y \to \# \mid Y \in N'\} \cup \{F' \to F'', \# \to \#\}, \\
R_2 &= \{F \to F, cF \to cF', g \to \#, cg \to cg', \# \to \#\} \\
&\quad \cup \{cX \to cxd_k g \mid (k : X \to x, \sigma(k), \varphi(k)) \in P'\} \\
&\quad \cup \{d_i \to \#, d''_i \to \#, ch \to chd''_i p_1 \mid i \in lab(P')\} \\
&\quad \cup \{cp_1 \to cp_2, cp_2 \to cp_3, cp_3 \to cp_4, cp_4 \to cp_5\}, \\
R'_1 &= \{(a, out)\},
\end{aligned}$$

$$\begin{aligned}R'_2 = &\{(F', out), (a, out), (p_5, out), (d_0, out; f, in)\}\\ &\cup \{(d_i, out; d'_j, in), (d''_i, out; d'_j, in) \mid i \in \sigma(j)\}\\ &\cup \{(Y, out; F'', in) \mid Y \in N'\}\\ &\cup \{(p_5, out; e_i, in), (Q, out; e_i, in) \mid (i : Q \to x, \sigma(i), \varphi(i)) \in P'\}.\end{aligned}$$

The system Π simulates the programmed grammar G' in the following way. The evolution rules, of the form $cX \to cxd_kg$, for $(k : X \to x, \sigma(k), \varphi(k)) \in P'$, present in region 2, simulate the context-free rules of G', while the symport/antiport rules are used to check if the simple evolution rules are applied according to the order defined by the labels in the success fields of G'. Moreover, the evolution rules of the form $ch \to chd''_ip_1$, for $i \in lab(P')$, present in region 2, are applied to skip a rule that cannot be applied and, in this case, the symport/antiport rules are used to be sure that the skipping of a rule has been done in a correct way (this mechanism simulates the appearance checking mechanism of the grammar G'). The evolution rules $F \to F, cF \to cF'$, together with the symport/antiport rules $(F', out), (Y, out; F'', in)$, for $Y \in N'$, are used to make sure that, when the computation halts, there is no symbol from N' in region 2. The catalyst c is used to ensure that in each step only one simple evolution rule is executed.

We pass now to show in more detail how a computation of Π works. After a simple evolution (catalytic) rule of the form

$$cZ \to czd_mg, \text{ for some } (m : Z \to z, \sigma(m), \varphi(m)) \in P',$$

has been applied in region 2, the objects d_m and g have been produced. The object d_m indicates that the rule of G' with label m has been applied, while the object g is used only to "keep busy" the catalyst c. After the object d_m has been produced, it must leave region 2 (because of the simple evolution rule that produces the trap symbol #). But the object d_m can exit only if in region 1 the "right" predecessor ("right" in the sense of the order defined by the success/failure fields of G') is present. In fact, to exit from region 2, the object d_m must use (one of) the antiport rules from R'_2 of the form

$$(d_m, out; d'_j, in), \text{ for } j \text{ such that } m \in \varphi(j).$$

When the object d_m goes in region 1, it is changed in d'_m and this object will store the new predecessor (this means that it stores the label of the last applied rule) and the computation will continue. If a rule cannot be applied, then we must simulate the appearance checking mechanism of the grammar. It is possible to skip the application of a rule by using the simple evolution rule

$$ch \to chd''_qp_1, \text{ for some } q \in lab(P'),$$

present in region 2. The application of this rule means that the context-free rule of G' with label q cannot be applied. We obtain the objects d''_q and p_1. The object d''_q must go to region 1, but, as in the previous case, it can pass to region 1 only if the "right" predecessor is stored in this region. In fact, in order to exit from region 2, the object d''_q must use (one of) the antiport rules associated with membrane 2,

$$(d''_q, out; d'_j, in), \text{ for } j \text{ such that } q \in \varphi(j).$$

When the object d''_q arrives in region 1, the evolution rule $d''_q \to d'_q e_q$, is applied, which creates the new predecessor d'_q and the object e_q that must check if the rule $(q : Q \to z, \sigma(q), \varphi(q))$ was skipped correctly.

After producing e_q, the antiport rule $(Q, out; e_q, in)$, associated with membrane 2 might be applied. If this happens, then the evolution rule $Q \to \#$, is applied in region 1 and the computation never halts (this means that the skipping of the rule with label q was not correct).

If this antiport rule is not applied, then the skipping of the rule was correct and, now, the only thing to do is to "clean" region 1 of the object e_q. To this aim we use the object p_5 (it will arrive in region 1 only after checking the correctness of skipping a rule) and the antiport rule $(p_5, out; e_q, in)$, associated with membrane 2. After removing the object e_q from region 1, the simulation of rules from G' can be repeated. The objects $p_1, \cdots, p_5, g, g'$ are used to "keep busy" the catalyst c and to send, at the right time, the object p_5 to region 1.

Each produced terminal is sent out by the symport rule associated to membranes 1, 2 and the system Π sends out exactly the strings of $L(G) = L(G')$, hence, in this case it generates the length set of $L(G)$. □

5 Universality with Non-cooperative Rules

In this section we prove that EC P systems are universal even when using only simple non-cooperative evolution rules – hence no catalyst –, and simple symport/antiport rules. The "price" to pay, in comparison to the previous result, is to use a membrane structure composed of three membranes. The idea of the proof is to simulate a programmed grammar with appearance checking and unconditional transfer. As before, we use the fact that a programmed grammar with appearance checking and unconditional transfer is able to generate the recursively enumerable one-letter languages.

Theorem 3. $NECP_3(1, 1, ncoo) = NRE$.

Proof. Consider a programmed grammar $G = (N, \{a\}, P, S)$ with appearance checking and with unconditional transfer. As in the previous theorems, we add to P the triple $(0 : U \to S, \sigma(0), \varphi(0))$ with $\sigma(0) = \varphi(0) = l(S)$ where U is a new non-terminal and $l(S) = \{k \in lab(P) \mid (k : S \to x, \sigma(k), \varphi(k)) \in P\}$. In this way, we obtain a new grammar, $G' = (N', \{a\}, P', U)$, with $N' = N \cup \{U\}$; clearly, $L(G) = L(G')$ and each derivation in G' starts with the rule with label 0.

For each $x \in N'$ we consider a new symbol $\bar{x}$, we denote by $\bar{N}$ the set of such symbols, and we define the morphism $\gamma : (N' \cup T)^* \longrightarrow (\bar{N} \cup T)^*$ by $\gamma(x) = \bar{x}$, if $x \in N'$, and $\gamma(x) = x$, if $x \in T$.

We construct the EC P system

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

with:

$$
\begin{aligned}
O &= N' \cup \bar{N} \cup \{d_i, d_i'', d_i''', e_i, e_i' \mid i \in lab(P')\} \\
&\quad \cup \{a, p_1 \cdots p_5, c, h, \bar{h}, c_1, c_2, \cdots, c_7, c', \#, F, F', f\}, \\
\mu &= [_1[_2[_3\ ]_3]_2]_1, \\
w_1 &= \emptyset, \\
w_2 &= FfUh, \\
w_3 &= c, \\
R_1 &= \{e_i \to e_i' \mid i \in lab(P')\} \cup \{Y \to \# \mid Y \in N'\}, \\
R_2 &= \{\bar{Y} \to Y \mid Y \in N'\} \cup \{\bar{h} \to h, F \to F, c \to c_1, c_1 \to c_2, \cdots, c_6 \to c_7\} \\
&\quad \cup \{d_i \to d_i', d_i'' \to d_i' e_i \mid i \in lab(P')\}, \\
R_3 &= \{X \to \gamma(x) d_k \mid (k : X \to x, \sigma(k), \varphi(k)) \in P'\} \\
&\quad \cup \{h \to \bar{h} d_i'' p_1, d_i \to \#, d_i'' \to \# \mid i \in lab(P')\} \\
&\quad \cup \{p_1 \to p_2, p_2 \to p_3, \cdots, p_4 \to p_5, c_7 \to c, c \to c', F \to F'\}, \\
R_1' &= \{(a, out)\}, \\
R_2' &= \{(Q, out; e_i', in), (e_i, out), (p_5, out; e_i', in) \mid (i : Q \to z, \sigma(i), \varphi(i)) \in P'\} \\
&\quad \cup \{(F', out), (a, out)\} \cup \{(Y, out; F', in) \mid Y \in N'\}, \\
R_3' &= \{(\bar{Y}, out), (c, out; Y, in) \mid Y \in N'\} \\
&\quad \cup \{(\bar{h}, out), (F', out), (c', out; F, in), (c, out; h, in), (c_7, in), (a, out)\} \\
&\quad \cup \{(d_i, out; d_j', in), (d_i'', out; d_j', in) \mid i \in \sigma(j)\}.
\end{aligned}
$$

The system Π simulates the programmed grammar G' in the following way. In region 3 (the inner one) we simulate the application of the context-free rules of G', using the simple evolution rules

$$X \to \gamma(x) d_k, \text{ for } (k : X \to x, \sigma(k), \varphi(k)) \in P',$$

while the symport/antiport rules

$$(d_i, out; d_j', in), (d_i'', out; d_j', in), \text{ for } i \text{ and } j \text{ such that } i \in \sigma(j),$$

associated with membrane 3, take care of the application in the correct order of the rules of G'.

During the simulation, using the symport/antiport rules

$$(c, out; Y, in), \text{ for } Y \in N',$$

associated with membrane 3, we move from region 2 to region 3 the non-terminal that must be rewritten, and after performing such a rewriting, we store in region 2 the objects obtained. Finally, we use region 1 to implement the appearance checking mechanism of G', using the antiport rules

$$(Q, out; e_i', in), \text{ for } (i : Q \to z, \sigma(i), \varphi(i)) \in P',$$

associated with membrane 2.

The evolution rules $F \to F, F \to F'$, present in region 2 and 3 respectively, together with the antiport rule $(c', out; F, in)$ associated with membrane 3, and with the symport rule (F', out) associated with membranes 3 and 2, are used to make sure that, when the computation halts, there are no objects from N' in region 2.

Now we pass to describe in more details the work of the system Π.

After applying an antiport rule

$$(c, out; Z, in), \text{ for some } Z \in N',$$

associated with membrane 3, the object Z arrives in region 3 and the evolution rule

$$Z \to \gamma(z)d_m, \text{ with } (m : Z \to z, \sigma(m), \varphi(m)) \in P',$$

is applied. The non-terminal objects of $\gamma(z)$ are sent to region 2, using the symport rules

$$(\bar{Y}, out), \text{ for } Y \in N',$$

associated with membrane 3. The object d_m indicates that the rule of G' with label m has been applied; as soon as such an object has been produced it must leave region 3 (because of the evolution rule that produces the trap symbol #). The object d_m can exit only if in region 2 there exists the "right" predecessor ("right" in the sense of the order defined by the success/failure fields of G'). In fact, to leave region 3 the object d_m must use (one of) the antiport rules associated with membrane 3 of the form

$$(d_m, out; d'_j, in), \text{ for } m \in \sigma(j).$$

Such an antiport rule can be applied only if in region 2 one of the predecessors of d_m is present. When the object d_m goes in region 2, it is changed in d'_m and this object will store the new predecessor and the computation continues.

If a rule cannot be applied, then we must simulate the appearance checking mechanism of the grammar: we introduce in region 3 the special object h using the antiport rule $(c, out; h, in)$, associated with membrane 3, and, after that, we can apply the simple evolution rule

$$h \to \bar{h}d''_q p_1, \text{ for some } q \in lab(P'),$$

present in region 3. The application of this rule means that the context-free rule of G' with label q cannot be applied. The application of this evolution rule produces the objects d''_q, $\bar{h}$, and p_1. The object $\bar{h}$ is immediately sent out to region 2 and the object d''_q must go to region 2, but, as in the previous case, it can pass to region 2 only if the "right" predecessor is there. In fact, to exit from region 3, the object d''_q must use (one of) the antiport rules associated with membrane 3

$$(d''_q, out; d'_j, in), \text{ for } q \in \sigma(j).$$

In this case, when the object d''_q arrives in region 2, the evolution rule $d''_q \to d'_q e_q$, is applied and it creates the new predecessor d'_q and the object e_q that will check if the rule with label q was skipped correctly. After producing e_q, this object is sent out to region 1, using the symport rule (e_q, out), associated with membrane 2. As soon as the object e_q arrives in region 1, it is changed to e'_q and, at the next step, the antiport rule

$$(Q, out; e'_q, in), \text{ for } (q : Q \to z, \sigma(q), \varphi(q)) \in P',$$

associated with membrane 2, might be applied. If this happens, then the computation never halts, because the evolution rule $Q \to \#$, present in region 1, is applied (this means that the skipping of the rule with label q was not correct). If this antiport rule is not applied, then the skipping of the rule was correct and, now, the only thing to do is the cleaning of region 1 of the object e'_q. To this aim we use the object p_5 (it will arrive in region 2 only after checking the correctness of skipping a rule), together with the antiport rule

$$(p_5, out; e'_q, in),$$

associated with membrane 2. After removing the object e'_q from region 1, the simulation of the rules from G' can be repeated. Finally, we observe that the objects $p_1, p_2, \cdots, p_5$ are used to send at the "right" time the object p_5 to region 1, and the objects $c_1, c_2, \cdots, c_7$ are used to "keep busy" the special object c.

In each step, each produced terminal is sent out by the symport rules associated with membrane 1, 2, and 3, and we collect the output in the enviroment. Thus, the system Π sends out exactly the strings of $L(G) = L(G')$, and, hence, it generates the length set of $L(G)$. □

6 Open Problems

We have studied here the evolution-communication P systems with what we called the *mixed approach*, which is only one (probably the less restrictive) among the possible ways to define a computation in an EC P system. We can have, at least, two other approaches that we describe here, rather informally; these two approaches might be seen as inspired, in some extent, from biology:

(i) the *evolutive approach*:
the simple evolution rules of the system have priority over the symport/antiport rules (in *biological* terms: an organism evolves by itself as much as possible (*evolution*), until it needs something from others organisms (*communication*));

(ii) the *communicative approach*:
the symport/antiport rules of the system have priority over the simple evolution rules (in *biological* terms: an organism tries to *communicate* with the others organisms, and only when it cannot comunicate anymore, then it starts (or continue) its *evolution*).

We consider interesting the study of these (and possibly other) different approaches to have computation in an EC P system. For instance, a general question of interest could be to check what is *more important*, in this framework, evolution or communication.

Moreover, we want point to out that we have given two universality results for the EC P systems proving the equivalence with NRE. What can we say about the equivalence with $PsRE$? Are similar results true?

Finally, we consider of interest to study the relations between the power of simple evolution rules and the power of symport/antiport rules. In other words, is there some kind of trade-off between these two types of rules?

7 Final Remarks

We have introduced a new variant of P systems called evolution-communication P system, that simply joins classical evolution rules (without communication targets) and communication rules (symport/antiport rules). This new model seems to be closer to the biology, because it does not use target indications, but simple symport/antiport rules to realize the communication. We have presented a particular way of defining a computation in such a system, called the *mixed approach*, and we have proven that the EC P systems are universal when using catalytic rules (with only one catalyst), and symport/antiport rules of weight one, with a membrane structure consisting of two membranes; moreover, we have proven that, if we use three membranes, we can have universality also without catalysts (hence with non-cooperative rules) and, again, using symport/antiport rules of weight one. We can say that the results presented in this paper can stay in the middle between the results presented in [5] and [6] on P systems with evolution rules (or catalytic rules) and with target indications, and the results presented in [4] on P systems with symport/antiport rules. Finally we have suggested some different strategies (*evolutive and communicative*) to define a computation in this new variant; also, several open problems are presented.

References

1. F. Bernardini, V. Manca, P systems with boundary rules, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 97–101.
2. J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. H. Fernau, F. Stephan: How powerful is unconditional transfer? When UT meets AC, *Proceedings of the 3rd International Conference on Developments in Language Theory* (S. Bozapalidis, ed.), 1997, 249–260.
4. P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 237–248.
5. G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002.
6. P. Sosik, P systems versus register machines: two universality proofs, *Pre-proceedings of Workshop on Membrane Computing, WMC-CdeA2002* (G. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, 371–382.