Triangle: Language, Literature, Computation, n. 6, 2011
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

On Promoters/Inhibitors and Symport/Antiport with Traces in P Systems

Mihai Ionescu*

University of Pitesti
Faculty of Mathematics and Computer Science
Pitesti, Arges, Romania
E-mail: armandmihai.ionescu@gmail.com

Summary. This article brings together some rather powerful results on P systems in which the computation is performed by the communication of objects through symport and antiport rules considering the trace of an object through membranes, on the one hand, and by P systems with object-rewriting non-cooperative rules, promoters/inhibitors at the level of rules and only one catalyst, on the other. It is recalled here that computational universality can be reached whit these formalisms and that some of the proofs can be sketched. Three ideas are also put forward to brake the direct relationship (infinite hierarchy) induced by the size of the considered alphabet and the number of the membranes needed in a P system (with traces) to generate recursively enumerable languages on the chosen alphabet.

1 Introduction

The present study focuses on P systems with a purely communicative functioning from two different perspectives: the “classical" one, in which the

* This paper was possible thanks to the research grant “Programa Nacional para la Formación del Profesorado Universitario” from the Spanish Ministry of Education, Culture and Sport and to CNCSIS grant RP-4 12/01.07.2009

result of the computation is the (number of) objects collected in a specified membrane, as introduced in [6]; and a "non-classical" one in which the result of the computation is a *trace* of a certain object (that is, the string of labels of the membranes visited by this object, as introduced in [20]).

Symport/antiport phenomena are inspired by the biological process in which two molecules pass together, simultaneously, through a membrane, in the same direction (symport), or in opposite directions (antiport). For further biochemical details the reader is asked to see [1] and [2]. Technically, the rules used in P systems as models of these biological processes are of the form (x, in) and (x, out), as models of symport, and $(x, out; y, in)$ as a model of antiport, where x, y are strings of symbols representing multisets of chemicals. Of course, this is a generalization of what happens in biology, where mainly pairs of chemicals are coupled. Several classes of P systems of this type were considered in [4], [6], [7], etc.

The other formalism we recall here is the one of P systems with promoted/inhibited rules, which also has a strong biological motivation. More precisely, a promoter is a chemical within a living cell which makes a reaction happen only in its presence. The inhibitor is the opposite: the reaction can not take place if a certain chemical is present in the cell. These biological considerations have been formalized in [12] as $u \to v|_a$(u evolves to v in the presence of the promoter a), and $u \to v|_{\neg b}$ (u cannot evolve to v if the inhibitor b is present in the same membrane region).

For the reader's convenience, we recall the fact that P systems are distributed parallel computing models which abstract from the structure and the functioning of the living cells. In short, we have a *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* where multisets of certain *objects* are placed (Figure 1 illustrates these notions); the objects evolve according to given *evolution rules*, which are applied nondeterministically (the rules to be used and the objects to evolve are randomly chosen) in a maximally parallel manner (in each step, all objects which can evolve must). The objects can also be communicated from one region to another. In this way, we get *transitions* from one *configuration* of the system to the next. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects from a specified *output membrane*.

Details can be found at the web address `http://bioinformatics.bio.disco.unimib.it/psystems`.

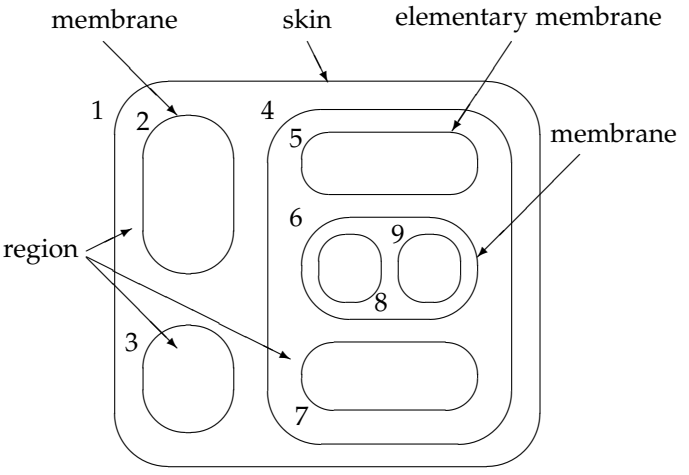

Figure 1: A membrane structure

One particular case of the abovementioned evolution rules is that of catalytic rules, which model the biological reactions that can take place only with the help of certain enzymatic proteins (which participate in reactions and remain unmodified after they occur). Another important type is promoted/inhibited reactions, which take place in the presence/absence of certain chemicals that are not directly involved in reactions.

In this abstract, symbolic, mathematical framework it is interesting to see what the computational power is when “low" cooperation features are used. In this regard, as was shown in [14], P systems with non-cooperative and catalytic rules and only two distinct catalysts are computational universal. Also, in [12] a model with non-cooperative rules, one catalyst and promoters at the level of rules is shown to be universal.

In this paper we emphasize the computational power of the systems with non-cooperative rules, catalytic rules with one catalyst and promoters/inhibitors in both generative and accepting cases. We will also mention the latest results regarding the computational power of the “traced" variant of P systems.

2 Preliminaries

The language theory notions we use here are standard, and can be found, for instance, in [10], [11]. We only mention that we denote by V^* the free monoid generated by an alphabet V; λ is the empty string, $|x|$ is the length of $x \in V^*$, and $|x|_a$ is the number of occurrences of the symbol $a \in V$ in the string $x \in V^*$. For $x \in V^*$ we denote $alph(x) = \{a \in V \mid |x|_a \geq 1\}$ (the set of symbols appearing in x), and for a language $L \subseteq V^*$ we write $alph(L) = \bigcup_{x \in L} alph(x)$. By REG, CF, CS, REC, RE we denote the families of regular, context-free, context-sensitive, recursive, and recursively enumerable languages, respectively.

In the (sketches of the) proofs of the theorems presented here, we used tools such as Regulated Rewriting (more precisely the fact that the families of languages generated by regularly controlled context-free grammars with appearance checking and erasing rules are equal in generative power to the family of all recursively enumerable languages over the same alphabet), and the generative power of Register Machines (which are computationally universal). Those readers who are familiar with these concepts can skip the first two subsections.

2.1 Regulated rewriting

In any Chomsky grammar, at some given step in a derivation one can use for rewriting any applicable rule in any desired place of the sentential form. In order to restrict this nondeterminism some regulating mechanisms, which can control the derivation process, were considered. Using such regulations we can reach computational universality even if we use context-free grammars as a core generative device. In the literature there are many types of regulations which restrict the use of rules in a Chomsky grammar (see [3], [18]). Here we will present only regularly controlled grammars with appearance checking and λ–rules.

A *regularly controlled context-free grammar with appearance checking* is a 6-tuple $G_{rC} = (N, T, P, S, R, F)$ where N,T,P, and S are specified as in context-free grammars, R is a regular language over P, and F is a subset of P.

For a rule $p = A \to w \in P$ and $x, y \in V_G^*$ we write $x \Longrightarrow_p^{ac} y$ if either

1. $x = x_1 A x_2$ and $y = x_1 w x_2$, or
2. $x = y$, A does not appear in x, and $p \in F$.

The language $L(G)$ generated by G with appearance checking consists of all words $w \in T^*$ such that there is a derivation

$$S \Longrightarrow_{p_1}^{ac} w_1 \Longrightarrow_{p_2}^{ac} w_2 \cdots \Longrightarrow_{p_n}^{ac} w_n = w$$

with $p_1 p_2 \cdots p_n \in R$.

By $\mathcal{L}(\lambda rC_{ac})$ we denote the families of languages generated by regularly controlled grammars with appearance checking and erasing rules. The following result stands:

$$\mathcal{L}(\lambda rC_{ac}) = \mathcal{L}(RE)$$

where by $\mathcal{L}(RE)$ we denote the family of all recursively enumerable languages over the same alphabet T.

2.2 Register machines

The power of Minsky's register machine [16] was also used in some of the proofs of the theorems that follow, which is why we recall this notion here. This machine runs a program consisting of numbered instructions of several simple types. Several variants of register machines with a different number of registers and different instructions sets were shown to be computationally universal (see [16] for some original definitions and [15] for the definition we use here).

A *n-register machine* is a construct $M = (n, P, i, h)$, where:

- n is the number of registers,
- P is a set of labeled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M, and j, k, l are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),
- i is the initial label, and
- h is the final label.

The machine is capable of the following instructions:

$(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.

$(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.

halt : This instruction stops the machine. This additional instruction can only be assigned to the final label h.

A deterministic m-register machine can analyze an input $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to α, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty(this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

2.3 P systems prerequisites

A P system (of degree $m \geq 1$) with symbol–objects and rewriting evolution rules is a construct

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_0),$$

where:

- V is the alphabet of Π; its elements are called *objects*;
- $C \subseteq V$ is the set of catalysts;
- μ is a membrane structure consisting of m membranes labeled $1, 2, \cdots, m$;
- w_i, $1 \leq i \leq m$, specify the multisets of objects present in the corresponding regions i at the beginning of a computation;
- R_i, $1 \leq i \leq m$, are finite sets of evolution rules over V associated with the regions $1, 2, \ldots, m$ of μ, and ρ_i is a partial order relation over R_i (a priority relation); these evolution rules are of the form $a \to v$ or $ca \to cv$, where a is an object from $V - C$ and v is a string over

  $$(V - C) \times (\{here, out, in\})$$

  (In general, the target indications *here*, *out*, *in* are written as subscripts of objects from V.);
- i_0 is a number between 0 and m and specifies the output membrane of Π (in case of 0, the environment is used for the output).

Starting from the original model of P system, several variants were proposed (see [17]). One of them is *P systems with promoters/inhibitors* and was introduced, as mentioned previously, in [12]. In the case of promoters, the rules (reactions) are possible only in the presence of certain symbols. An object a is a *promoter* for a rule $u \to v$, and we denote this by $u \to v|_a$, if the rule is active only in the presence of object a. An object b is an *inhibitor* for

a rule $u \to v$, and we denote this by $u \to v|_{\neg b}$, if the rule is active only if inhibitor b is not present in the region. In particular, promoters/inhibitors themselves can evolve according to some rules.

The difference between catalysts and promoters consists of the fact that the catalysts directly participate in rules (but are not modified by them), and they are counted as any other objects, so that the number of applications of a rule is as big as the number of copies of the catalyst, while in the case of promoters, the presence of the promoter objects makes it possible to use the associated rule as many times as possible, without any restriction; moreover, the promoting objects do not necessarily directly participate in the rules. As a consequence, it can be seen that the catalysts inhibit the parallelism of the system while the promoters/inhibitors only guide the computation process.

The P system with the mentioned features starts to evolve from the initial configuration, as in the classical P system, to the final configuration. The result of the halting computation is the number of objects present in the region i_0 in the halting configuration. The set of all numbers constructed in this way by a system Π is denoted by $N(\Pi)$. For this kind of P systems we will use the following notation:

$$NOP_m(\alpha, \beta), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\}, \beta \in \{proR, inhR\}$$

to denote the family of sets of natural numbers generated by P systems with at most m membranes, evolution rules that can be non-cooperative $(ncoo)$, cooperative (coo), or catalytic (cat_k), using at most k catalysts, and promoters $(proR)$ or inhibitors $(inhR)$ at the level of rules.

We may also consider the vector $\Psi(w)$ as the result of the halting computation (the vector of multiplicities of objects) where w is the multiset present in the region i_0 in the halting configuration. In this case, the set of all vectors constructed in this way by a system Π is denoted by $Ps(\Pi)$. We will also use the following notation:

$$PsIP_m(\alpha, \beta), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\}, \beta \in \{proR, inhR\},$$

to denote the family of sets of vectors of natural numbers accepted by P systems with at most m membranes, evolution rules that can be non-cooperative $(ncoo)$, cooperative (coo), or catalytic (cat_k), using at most k catalysts, and promoters $(proR)$ or inhibitors $(inhR)$ at the level of rules. Here, I stands for P systems with internal input. For generative devices we will write $PsP_m(\cdots, \cdots)$.

In this paper we will recall how the regularly regulated context-free grammars with appearance checking can be used to prove the computational universality of this type of P systems.We will also present the deterministic P systems that accept sets of vectors of natural numbers.

We also recall here the case in which the trace of a certain object is considered, so we have P systems of the following form:

$$\Pi = (V, t, T, h, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m),$$

where V is an alphabet, $t \in V$ (a distinguished object, "the traveler"), T is an alphabet, $h : \{1, 2, \ldots, m\} \longrightarrow T \cup \{\lambda\}$ is a weak coding, $w_1, \ldots, w_m$ are strings representing the multisets of objects present in the m regions of μ, E is the set of objects present in arbitrarily many copies in the environment, and $R_1, \ldots, R_m$ are the sets of symport and antiport rules (with promoters or inhibitors) associated with the m membranes. The traveler is present in exactly one copy in the system, that is, $|w_1 \ldots w_m|_t = 1$ and $t \notin E$.

Let $\sigma = C_1 C_2 \ldots C_k, k \geq 1$, be a halting computation with respect to Π, with $C_1 = (w_1, \ldots, w_m, \lambda)$ the initial configuration, and $C_i = (z_1^{(i)}, \ldots, z_m^{(i)}, z_e^{(i)})$ the configuration at step $i, 1 \leq i \leq k$. If $|z_j^{(i)}|_t = 1$ for some $1 \leq j \leq m$, then we write $C_i(t) = j$ (therefore, $C_i(t)$ is the label of the membrane where t is placed). If $|z_j^{(i)}|_t = 0$ for all $j = 1, 2, \ldots, m$, then we put $C_i(t) = \lambda$. Then, *the trace* of t in the computation σ is

$$trace(t, \sigma) = C_1(t) C_2(t) \ldots C_k(t).$$

The computation σ is said to generate the string $h(trace(t, \sigma))$. Hence the language generated by Π is $L(\Pi) = \{h(trace(t, \sigma)) \mid \sigma$ is a halting computation in $\Pi\}$.

We denote by $LP_m(psym_p, panti_q)$ the family of languages generated by P systems with at most m membranes, with symport rules of weight at most p and antiport rules of weight at most q, using promoters; when the rules have associated forbidding contexts we write *fsym, fanti* instead of *psym, panti*; when the rules are used in the free mode (they have no promoter/inhibitor symbols associated), we remove the initial "p" and "f" from *psym, panti* and *fsym, fanti*. As usual, the subscript m is replaced by $*$ when no bound on the number of used membranes is considered; similarly, if we use symport or antiport rules of an arbitrary weight, then the subscripts p, q are replaced with $*$.

3 Some relevant examples

In this section we will present some examples of P systems computing some "sensitive" tasks using the types of P systems discussed above. First we will construct a P system with promoters that, having as input two values, say 0 and/or 1, computes the *and* operation (see Figure 1).

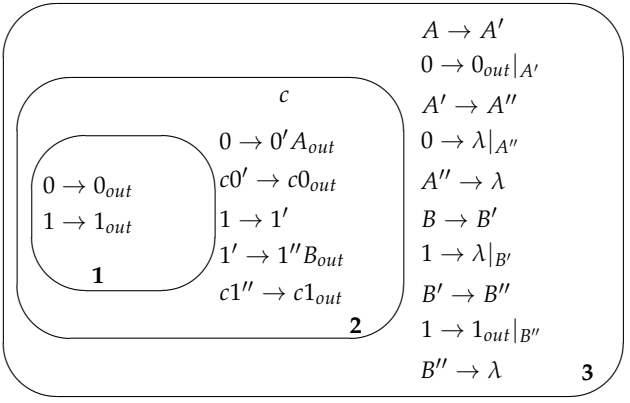

Fig. 1. Simulation of the AND gate using promoters and one catalyst

Formally, we define the following P system

$$\Pi_{AND} = (V, C, \mu, w_1, w_2, w_3, R_1, R_2, R_3, 0),$$

where:

- $V = \{0, 1, 0', 1', 1'', A, A', A'', B, B', B'', c\}$;
- $C = \{c\}$;
- $\mu = [_3[_2[_1\ ]_1]_2]_3$;
- $w_1 = w_3 = \emptyset$, $w_2 = \{c\}$;
- $R_1 = \{1 \to 1_{out}, 0 \to 0_{out}\}$;
  $R_2 = \{0 \to 0'A_{out}, c0' \to c0_{out}, 1' \to 1''B_{out}, 1 \to 1',$
  $c1'' \to c1_{out}\}$;
  $R_3 = \{A \to A', 0 \to 0_{out}|_{A'}, A' \to A'', 0 \to \lambda|_{A''},$
  $A'' \to \lambda, B \to B', 1 \to \lambda|_{B'}, B' \to B'',$
  $1 \to 1_{out}|_{B''}, B'' \to \lambda\}$.

The simulation of the AND gate uses the catalyst c to inhibit the parallelism and to separate the entrance time of objects 0 and 1 into region 3. Depending on the entrance time, objects will be either deleted, or sent out into the environment. More specifically, if we consider that initially we had two objects 0 inside region 2, the rule $0 \to 0'A_{out}$ is executed. Its role is to introduce object A into region 3 to set up the "right" configuration of the region. Next, in region 2 the only applicable rule is $c0' \to c0_{out}$, which will introduce one object 0 into region 3. At the same time, in region 3 the rule $A \to A'$ is executed. Now, we will have in region 3 the objects A' and 0, and the rules that will be applied are $0 \to 0_{out}|_{A'}$ and $A' \to A''$. These rules guarantee that an object 0 is sent out into the environment. In the meantime, in region 2, the remaining object $0'$ reacts with the catalyst c and an object 0 will be introduced into region 3 (the rule used is again $c0' \to c0_{out}$). Here, object 0 will find a different context since now, in region 3 there is no object A'. Therefore, the rules $0 \to \lambda|_{A''}$ and $A'' \to \lambda$ are applied. Hence the initial configuration of the system is restored. Basically, a similar method stands for the other cases, with some minor changes: objects 1 enter into region 3 with one computational delay (because of the rule $1 \to 1'$ present in region 2) in order not to influence the processes executing in region 3; the first object 1 that enters into region 3 is deleted (as opposed to the above case when the first object 0 that arrives in region 3 is sent out) by using the rule $1 \to \lambda|_{B'}$.

Recall that membrane 1 can be entirely avoided, its role being only to specify the entry point of the input. Also, the result of computation is sent out into the environment even if it is actually obtained in region 3. These features are useful when we want to connect gates to circuits (see [13] for more details).

The second example (see Figure 2) uses context-free rules, inhibitors and one catalyst to compute the arithmetic difference between the initial multiplicity of two distinct objects, present at the beginning of computation into an input region.

Formally, we define the following P system

$$\Pi_{dif} = (V, C, \mu, w_1, w_2, R_1, R_2, 2),$$

where:

- $V = \{a, b, a', b', d, A, B, c\}$;
- $C = \{c\}$;

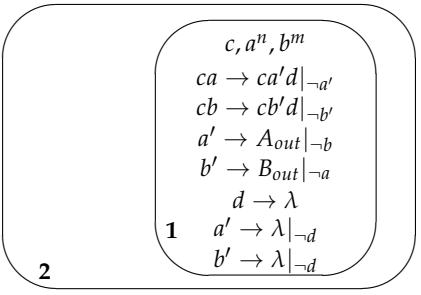

Fig. 2. Integer subtraction using inhibitors and one catalyst

- $\mu = [_2[_1\ ]_1]_2$;
- $w_1 = \{c, a^n, b^m\}$, $w_2 = \emptyset$;
- $R_1 = \{ca \to ca'd|_{\neg a'}, cb \to cb'd|_{\neg b'}, a' \to A_{out}|_{\neg b},$
  $b' \to B_{out}|_{\neg a}, d \to \lambda, a' \to \lambda|_{\neg d}, b' \to \lambda|_{\neg d}\}$;
  $R_2 = \emptyset$.

The system starts the computation by inputting into membrane 1 a catalyst c and the objects a^n, b^n, whose multiplicity we want to subtract. The result of computation is sent to region 2 and it is represented by:

- A^{n-m} if $n > m$;
- B^{m-n} if $m > n$;
- no object is sent to region 2 meaning that $m = n$.

We will not give many details regarding the computations here (these can be found in [19]). We only mention that the system works as follows: while there are still objects a and b, they are deleted in pairs, iteratively, until there are no more objects a, for instance (or objects b). At that moment, the flow of computation changes and as a result, also iteratively, the remaining objects b (or objects a, respectively) are sent out. During the computation, the promoters control the derivation process, while the catalyst inhibits the parallelism.

The third example (detailed also in [20]) shows how a P system considering traces of an object is computing. Consider the system

$$\Pi = (\{d, t\}, t, \{a, b, c\}, h, [\ 1[\ 2[\ 3[\ 4[\ 5\ ]_5\ ]_4\ ]_3\ ]_2\ ]_1, t,$$
$$\emptyset, \emptyset, \emptyset, \emptyset, \{d\}, R_1, R_2, R_3, R_4, R_5),$$

with $h(1) = a, h(3) = b, h(5) = c, h(2) = h(4) = \lambda$, and the following sets of rules:

$$R_1 = R_3 = R_5 = \{(t, out),\ (td, in)\},$$
$$R_2 = R_4 = \{(t, in),\ (d, in)\}.$$

First, the traveler brings $n \geq 1$ copies of d from the environment (each of them immediately enters membrane 2), and then the traveler goes to membrane 2. Subsequently the traveler brings $m \leq n$ copies of d into membrane 3 (each of them immediately membrane 4), and then the traveler goes to membrane 4 (at this moment, it is possible that some copies of d remain in membrane 2). From membrane 4, all copies of d are carried into membrane 5; the computation stops with the traveler in membrane 4. Thus, for any computation σ of this type, we have

$$trace(t, \sigma) = 1^{n+1}(23)^m(45)^m 4, \text{ for some } n \geq 1, m \leq n.$$

The traveler can also end up in membrane 2, after introducing all copies of d in membrane 3, and returning to membrane 2. In the case of such a computation σ we have

$$trace(t, \sigma) = 1^{n+1}(23)^n 2, \text{ for some } n \geq 1.$$

Finally, we can also have the trivial computation where t enters only membrane 2, without any copy of d present in the system, and this leads to $trace(t, \sigma) = 12$.

Consequently,

$$L(\Pi) = \{a^{n+1}b^m c^m \mid n \geq 1, m \leq n\} \cup \{a^{n+1}b^n \mid n \geq 1\} \cup \{a\}.$$

Clearly, this language is not a context-free one. Note that system Π only has symport rules, and that the rules are freely applied (we use no promoter or inhibitor).

4 Universality results - using P systems with promoted/inhibited rules

4.1 Computational universality – the generative case

We present two universality results concerning P systems with promoters or inhibitors at the level of rules. Both proofs are based on the simulations of

regularly controlled context-free grammars with appearance checking. Only the sketch of the first proof is given here. The detailed proofs can be found in [19].

As we presented in Subsection 2.1, the family of languages generated by such grammars $\mathcal{L}(\lambda rC_{ac})$ is equal to the family of all recursively enumerable languages $\mathcal{L}(RE)$. As a particular case, by NRE we denote the family of Turing computable sets of numbers. This family is isomorphic to the family of length sets of languages generated by regularly controlled context-free grammars with appearance checking over one letter terminal alphabet.

Recall also, that, because P systems with symbol-objects operate with multisets of objects (therefore we do not have the order given by strings) at most we can study the equivalence with a family of vectors of natural numbers.

We denote by $PsP_m(cat, proR)$, the family of sets of vectors of natural numbers computed by systems with at most m membranes, 1 catalyst (say c) and objects as promoters.

Theorem 1. $PsP_m(cat_1, proR) = PsRE.$

Proof. We will consider for this proof the implication $NRE \subseteq PsP_2(cat, proR)$; the other way around is a straightforward construction based on the Church-Turing thesis.

Let $G_{reg} = (N_{reg}, T_{reg}, P_{reg}, S_{reg})$ be a regular grammar generating the regular set L_{reg}. We denote by r the number of rules in P_{reg}. The rules of P_{reg} are enumerated as $i : (M_i \to p_iQ_i)$ or $i : (M_i \to p_i)$ with $1 \leq i \leq r$, where $M_i \in N_{reg}$ and $p_i \in T_{reg}$ $\forall$ $1 \leq i \leq r$. For any such grammar G_{reg} we can construct an equivalent right–linear grammar $G' = (N', T', P', S')$ in the following way:

$T' = T_{reg}$,
$S' = S_{reg}$,
$N' = N_{reg} \cup \{M_{(i,1)}, M_{(i,2)}, M_{(i,3)} \mid 1 \leq i \leq r\}$.

For any rule $i : (M_i \to p_iQ_i) \in P_{reg}$ or $i : (M_i \to p_i) \in P_{reg}$, $1 \leq i \leq r$ we will have in P' the sequence of rules:

$$M_i \to M_{(i,1)},\ M_{(i,1)} \to M_{(i,2)},\ M_{(i,2)} \to M_{(i,3)},\ M_{(i,3)} \to p_iQ_i,$$
$$M_i \to M_{(i,1)},\ M_{(i,1)} \to M_{(i,2)},\ M_{(i,2)} \to M_{(i,3)},\ M_{(i,3)} \to p_i$$

respectively. Moreover, P' does not contain other rules excepting the rules considered above.

In other words, the only difference between the two grammars is that a new terminal is produced in grammar G' after every fourth step of a derivation.

Now let us construct a P system which simulates the derivation process of a regularly controlled grammar with appearance checking. The system will use only two membranes, one catalyst and promoters. The innermost membrane will contain the generative mechanism and the results of computation will be sent out to the skin membrane which will be the output membrane of the system (the reason is that the catalyst is used during the computation to inhibit the parallelism and it cannot be removed, so we cannot obtain the number 0 as the result of computation if we use only one membrane). In what follows we will discuss only the rules in the innermost membrane since the skin membrane does not execute any task (its role is only to collect the objects obtained during computation).

The promoters will be generated by a mechanism like the one presented above (promoters will be actually terminal symbols from T' and, therefore, they will be generated every fourth step). They will permit the execution of "context-free" rules in the "right" order – the order given by the regular mechanism.

In order to correctly simulate the appearance checking mechanism we have to modify the rules in grammar G' such that we replace each rule of type $M_{(i,3)} \to p_iQ_i$ by rules of type: $M_{(i,3)} \to p_iQ_if$ or $M_{(i,3)} \to p_iQ_ia$ depending on how the object p_i indicates a rule from F (in the regularly controlled grammar definition, the set $F \subset P$ represents the appearance checking set of rules; we will use the object a to identify that a rule with the corresponding label p_i is in the appearance checking set; if not, we will produce object f in the rule). We will also consider the same construction for the rules in G' of type $M_{(i,3)} \to p_i$, i.e., $M_{(i,3)} \to p_if$ or $M_{(i,3)} \to p_ia$. This means that, in the definition of our P system, for the inner membrane, we will have rules of the following types:

- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_iQ_if$ if p_i is not a label in the appearance checking set;
- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_iQ_ia$ if p_i is a label in the appearance checking set;
- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_if$ if p_i is not a label in the appearance checking set;
- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_ia$ if p_i is a label in the appearance checking set.

For a context-free rule $(p : (A \to \alpha)) \in G_{CF}$ ($G_{CF} = (N_{CF}, T_{CF}, P_{CF}, S_{CF})$) we will have the following sequence of rules in our P system:

$cA \to cD\alpha|_p$,
$p \to p'$,
$p' \to \lambda|_D$,
$D \to \lambda$.

Here, without losing generality. We have considered that $\alpha \in (N \cup T_{out})^*$ which means that if we apply the rule $p : (A \to \alpha)$ we will send the terminal symbols to the output region (recall that we are interested only in the number of objects).

This sequence of rules stands for the case when the context-free rule $A \to \alpha$ can be applied and, therefore must be applied. In the case when the rule mentioned cannot be applied we have to decide if the promoter present indicates a rule with the label in the appearance checking set or not.

First, let us consider the case when the promoter is not a label in the appearance checking set. In this case, recall that we deal with the following sequences of productions (from the regular mechanism):

- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_i Q_i f$, or
- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_i f$.

As a result of applying these rules we will have in the inner region, among others, the objects p and f. Let us consider that, for this case, we have the rules:

$f \to f_1$,
$f_1 \to f_2$,
$p' \to \#|_{f_2}$,
$f_2 \to \lambda$,
$\# \to \#'$,
$\#' \to \#$.

The first two rules from this group are meant to delay the execution of the third rule because we are not "sure" if the rule $cA \to cD\alpha|_p$ is or is not applied.

With a construction similar to the one above we can solve the case when we deal with rules that have labels in the appearance checking set. This means that the rules to be applied are of the types:

- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_i Q_i a$, or
- $M_i \to M_{(i,1)}$, $M_{(i,1)} \to M_{(i,2)}$, $M_{(i,2)} \to M_{(i,3)}$, $M_{(i,3)} \to p_i a$.

Here the difference from the previous case is that the trap symbol # is not generated if the rule $cA \to cD\alpha|_p$ cannot be applied. We only have to delete the promoter p'. It may interfere in the next steps of computation if it is not deleted. The rules below state the fact that if a rule is in the appearance checking set and it can not be applied even if it is indicated by the regular mechanism, then it can be skipped.

$a \to a_1$,
$a_1 \to a_2$,
$p' \to \lambda|_{a_2}$,
$a_2 \to \lambda$.

Finally, the initial configuration of the P system is composed of the starting symbol S_{Reg} of the regulating mechanism, the starting symbol S_{CF} of the context-free mechanism and the catalyst c. The system will evolve in a maximally parallel manner and its behavior is controlled by the catalyst and promoters.

It can be seen that the descriptional complexity in terms of number of membranes, number of catalysts and promoters is the same as in the original proof in [12], but simulating a different computational universal mechanism. □

As a particular case we have the following result:

Corollary 4.1 $NOP_2(cat_1, proR) = NRE.$

As can be seen, the promoters combined with one catalyst are sufficient to prove the computational universal capabilities of the P systems when using only context-free object rewriting rules. Also, a similar result, but for inhibitors not promoters, stands.

Theorem 2. $PsP_2(cat_1, inhR) = PsRE.$

As a particular case we have the following result:

Corollary 4.2 $NOP_2(cat_1, inhR) = NRE.$

4.2 Computational universality – the accepting case

The following theorems illustrate the computational universality (in their accepting variants) of P systems with object rewriting non-cooperative rules and promoters/inhibitors at the level of rules. The systems we propose simulate the moves of deterministic register machines. Moreover, the obtained

P systems are also deterministic. As in the previous subsection, we give here only the proof sketch of the first theorem, for the details suggesting [19].

Theorem 3. $PsIP_2(cat_1, proR) = PsRE.$

Proof. In order to prove this assertion we will simulate a n–register machine $M = (n, P, i, h)$. At each time during the computation, the current contents of register j is represented by the multiplicity of the object a_j.

Formally, we define the P system

$$\Pi = (V, C, [_1\ [_2\ ]_2\ ]_1, w_1 = \emptyset, w_2, R_1 = \emptyset, R_2, 1),$$

where:

$V = \{a_j, A_j, S_j \mid 1 \leq j \leq n\} \cup \{F, T\} \cup \{e, e' \mid (e : add(j), f) \in P\} \cup$
$\{e, e', e'' \mid (e : sub(j), f, z) \in P\},$
$C = \{c\},$
$w_2 = \{c, e, a_j^{k_j}, 1 \leq j \leq n, k_j \in N\},$
and R_2 is defined as follows:

- for each instruction $(e : add(j), f) \in P$, we add to R_2 the rules:
  $e \to e' A_j$
  $c \to c a_j|_{A_j}$
  $A_j \to \lambda$
  $e' \to f$
- for each instruction $(e : sub(j), f, z) \in P$, we add to R_2 the rules:
  $e \to e' T S_j$
  $c a_j \to c F|_{S_j}$
  $S_j \to \lambda$
  $e' \to e''$
  $T \to T'$
  $e'' \to f|_F$
  $F \to \lambda$
  $T' \to T''$
  $e'' \to z|_{T''}$
  $T'' \to \lambda$
- for the instruction $(h : halt) \in P$, we add to R_2 the rules:
  $a_j \to \#|_h$, $1 \leq j \leq n$
  $h \to \lambda$
- the rule $\# \to \#$ is added to R_2,
- no other rules are added to R_2.

The system works as following. Initially the P system starts the computation whit the objects $a_1^{k_1}, \cdots, a_n^{k_n}$, the catalyst c and the label e of the first instruction of the register machine we want to simulate in its input region (region 2). The vector $(k_1, \cdots, k_n)$ represents the vector that has to be accepted by our P system. □

Theorem 4. $PsIP_2(cat_1, inhR) = PsRE$.

5 Universality results - using P systems with traces and symport/antiport rules

The initial results presented in [20] (with which universality could be obtained using antiport rules of an arbitrary weight, as well as promoters or inhibitors), were significantly improved in [21].

Before presenting the latter we should mention that by lRE we mean the family of recursively enumerable languages over alphabets of size l, and that $lLP_m(sym_j, anti_k)$ denotes the family of languages over alphabets of size l defined by traces of P systems with symport/antiport whit at most m membranes, symport of weight at most j, and antiport of weight at most k.

Here are the results presented in [21]:

$$lLP_{l+1}(sym_0, anti_2) = lRE$$

$$lLP_{l+1}(sym_3, anti_0) = lRE$$

$$lLP_{l+2}(sym_2, anti_0) = lRE$$

For details regarding the proof, the reader is asked to consult the above mentioned paper.

It can be seen that the number of membranes used to obtain universality is strictly dependent on the size of the chosen alphabet, so an infinite hierarchy is created. The question we try to answer in the next subsection (more precisely we suggest some answers to it), is: Can lRE be obtained with P systems in which the number of membranes does not depend on l?

5.1 Ways of obtaining lRE with the number of membranes not depending on l

Several travelers

The firs way of decreasing the number of membranes is to consider several travelers. Let us suppose that alphabet T (which was mentioned in section 2.3) has k components (travelers) and the considered P system contains m membranes.

In this case V consists of $k * m$ symbols $a_{i,j}$, where $1 \leq i \leq k$, and $1 \leq j \leq m$. Let $\sigma = C_1 C_2 \dots C_k$, $k \geq 1$, be a halting computation and let $C_i(T) = \{a_{ij} |\ t_i$ be in membrane $j\}$.

We consider $trace(T, \sigma) = \{w_1, w_2, \dots, w_k | w_i \in V^*, \Psi_V(w_i) = \Psi_V(C_i(T))\}$, i.e. w_i is a linear arrangement of $C_i(T)$ (more precisely, we allow here any permutation). Hence, the language defined by a P system with several travelers is given by: $LT(\Pi) = \{h(trace(T, \sigma)) |\ \sigma$ is a halting computation in $\Pi\}$.

We should also mention that the language $LT(\Pi)$ can be over $k * m$ symbols (provided by the k travelers and m membranes considered).

Using an inverse morphism

The second idea (invitation) we propose is to consider an inverse morphism. Let us suppose we have the following morphism $h : V^* \to U^*$. Then, the inverse morphism of h is defined as $h^{-1} : U^* \to 2^{V^*}$, where $h^{-1}(y) = \{x \in V^* |\ h(x) = y\}$, $y \in U^*$.

Now let us consider the following example: let us have two alphabets, $V = \{a_1, a_2, \dots, a_k\}$, $U = \{0, 1\}$ and the mapping h defined as above, where $h(a_i) = 0^i 1$. It is obvious that this mapping is injective; hence, $card(h^{-1}(y)) = 1$. We choose now a recursively enumerable language $L \subseteq V^*$ and we write it in the form $L = h^{-1}(h(L))$. It it obvious (from the way we defined h) that choosing L from kRE, $h(L) \in 2RE$.

Using now the results from [21] we conclude that:

$$h(L) \in 2LP_3(sym_0, anti_2),$$

and

$$h(L) \in 2LP_3(sym_3, anti_0).$$

Changing the labels of membranes

Our final proposed answer to the core question of this section is to consider the change of the labels of the membranes. The idea is "borrowed" from the P systems formalisms which deal with active membranes (details in [17]).

Our (symport) rules of the form (x, in), (x, out) can be rewritten as $x[\ ]_i \to [x]_i$, and $[x]_i \to [\ ]_i x$, respectively, while the rule $(x, out; y, in)$ (the case of antiport) can be rewritten as $y[x]_i \to [y]_i x$. Generalizing, we can consider that whenever an object enters or exits a membrane it can change its label. Thus, our rules will become: $x[\ ]_i \to [x]_j$, $[x]_i \to [\ ]_j x$, and $y[x]_i \to [y]_j x$, respectively.

Once this point has been reached another problem might arise: namely, the conflict with other labels. To be more precise, when label i changes to j, a different membrane may have the latter, which is of course, quite unwelcome. For this reason, we thought of three ways of avoiding this conflict. The first is the *sequential use of rules*. The second solution is to considera *coherent set of rules* of the form (i, j), all passing from i to j. And the last one is to use *at most* one rule which changes the label and, in parallel, other rules which do not change the label of the membranes. The reader is encouraged to try them.

6 Final remarks and future work

We have put together some powerful results which arise from the mechanisms of P systems with promoters/inhibitors, and symport/antiport considering the trace of a certain object through membranes.

We have also recalled here a new way of defining the result of a computation in a P system with symport and antiport rules: namely, taking into consideration the trace of a specified object through membranes during a computation. This idea is attractive because it leads to a string associated with a computation rather than a number. We believe this last idea is worth considering also for other classes of P systems. Of particular interest are systems whose computation is based entirely on communication, such as those involving carriers [5] and various classes of systems with symport/antiport rules. For instance, we can consider P systems with symport/antiport rules with other control mechanisms on the use of rules or with other arrangements of membranes – (e.g., P systems working on networks of elementary membranes [8]).

In the last section we proposed three different ways of solving the infinite hierarchy problem related to the direct relationship between the necessary number of the membranes in a P system needed to generate a *RE* language.

We also want to propose here that the relationship of P systems with traces and *Gauss codes* be studied. (Gauss codes are one of the oldest problems leading to a formal language). Here is how they are obtained: consider a planar closed curve with simple crossing points (i.e. it is not a tangent point and the curve crosses itself only once that point). Assign the numbers $1, 2, \ldots, n$ to the n crossing points of a given curve c. A sequence $x(c)$ containing exactly two occurrences of each i (i between 1 and n) and describing the passing of the curve through the crossing points is called a Gauss code. More details about Gauss codes can be found in [22].) It seems natural, in a way, to think that the traveler considered here will "jump" on a Gauss curve and follow its path, but so far no work has been done in this direction.

References

1. B. Alberts et al., *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.
2. I.I. Ardelean, The relevance of cell membranes for P systems. General aspects, *Fundamenta Informaticae*, 49, 1-3 (2002).
3. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
4. C. Martín-Vide, A. Păun, Gh. Păun, G. Rozenberg, Membrane systems with coupled transport: Universality and normal forms, submitted, 2002.
5. C. Martín-Vide, Gh. Păun, G. Rozenberg, Membrane systems with carriers, *Theoretical Computer Sci.*, 270 (2002), 779–796.
6. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, *New Generation Computers*, to appear.
7. A. Păun, Gh. Păun, A. Rodriguez-Paton, Further remarks on P systems with symport rules, *Ann. Univ. Al.I. Cuza, Iaşi*, to appear.
8. A. Păun, Gh. Păun, G. Rozenberg, Computing by communication in networks of membranes, submitted, 2001.
9. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).
10. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.
11. A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

12. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg, Membrane Systems with Promoters/ Inhibitors *Acta Informatica*, 38, 10 (2002), 695-720.
13. R. Ceterchi, D. Sburlan, Simulating Boolean Circuits with P Systems, *Workshop on Membrane Computing WMC-Tarragona 2003*, (A. Alhazov, C. Martín-Vide, G. Păun, eds), TR 28/03, URV Tarragona, 2003.
14. R. Freund, L. Kari, M. Oswald, P. Sosik, Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient; submitted 2003.
15. S. Khrisna, A. Păun, Three Universality Results on P Systems, *Workshop on Membrane Computing WMC-Tarragona 2003*, (A. Alhazov, C. Martín-Vide, G. Păun, eds), TR 28/03, URV Tarragona, 2003, 198-206.
16. M.L. Minsky, *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, 1967.
17. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
18. Gh. Păun, G. Rozenberg, A Guide to Membrane Computing, *Theoretical Computer Science*, 287, 1 (2002), 73-100.
19. M. Ionescu, D. Sburlan, On P Systems with Symport/Antiport, *Journal of Universal Computer Science*, 10(5), 2004, 581-600.
20. M. Ionescu, C. Martín-Vide, G. Păun, P Systems with Symport/Antiport: The Trace of Objects, *Grammars*, 5(2), 2002, 65-70.
21. P. Frisco, H.J. Hoogeboom, Simulating Counter Automata by P Systems with Symport/Antiport, *Lecture Notes in Computer Science*, 2597, 2002, 288-301.
22. C. Martín-Vide, V. Mitrana, Gh. Păun (eds.), *Formal Languages and Applications*, Springer-Verlag, 2004, 20-21.