New Generation Computing, 20(2002)295-305
Ohmsha, Ltd. and Springer-Verlag

The Power of Communication: P Systems with Symport/Antiport

Andrei PĂUN
Department of Computer Science, University of Western Ontario
London, Ontario, Canada N6A 5B7
apaun@csd.uwo.ca

Gheorghe PĂUN
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
gpaun@imar.ro, gp@astor.urv.es

Received 8 March 2002
Revised manuscript received 1 February 2002

Abstract In the attempt to have a framework where the computation is done by communication only, we consider the biological phenomenon of trans-membrane transport of couples of chemicals (one say symport when two chemicals pass together through a membrane, in the same direction, and antiport when two chemicals pass simultaneously through a membrane, in opposite directions). Surprisingly enough, membrane systems without changing (evolving) the used objects and with the communication based on rules of this type are computationally complete, and this result is achieved even for pairs of communicated objects (as encountered in biology). Five membranes are used; the number of membranes is reduced to two if more than two chemicals may collaborate when passing through membranes.

Keywords: Molecular Computing, Membrane Computing, Symport, Antiport, Computational Universality.

§1 Introduction

P systems are distributed parallel computing models which start from the observation that the processes which take place in the complex structure of a living cell can be interpreted as a computation. The basic ingredients are a *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* (the space between a membrane and all directly inner membranes, if any inner membrane exists; Figure 1 illustrates

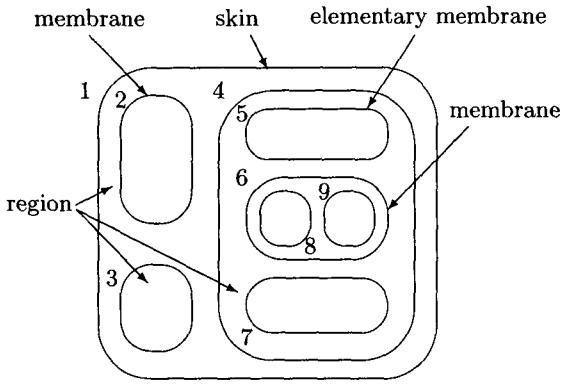


Fig. 1 A Membrane Structure

these notions) where multisets of certain *objects* are placed; the objects evolve according to given *evolution rules*, which are applied non-deterministically (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 do it). The objects can also be communicated from a region to another one. In this way, we get *transitions* between the *configurations* of the system. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects in a specified *output membrane*.

Since these computing devices were introduced[4)] several classes of P systems have been considered. Many of them were proved to be computationally complete, able to compute all Turing computable sets of natural numbers. When membrane division, or membrane creation is allowed, NP-complete problems are shown to be solved in linear time. Details can be found at the web address `http://bioinformatics.bio.disco.unimib.it/psystems`.

The communication of objects through membranes is one of the most important ingredients of a P system, and this led to the question to closely investigate the power of communication, to consider "purely communicative" systems, where the objects are not changed during a computation, but they just change their place with respect to the compartments of the system. A first attempt to solve this problem was done in Reference[3)] by considering certain "vehicle-objects" (a model of plasmids, or of vectors from gene cloning) which carry other objects through membranes. Here we follow another biochemical idea, that of membrane transport based on pairs of chemicals. When two chemicals can pass through a membrane only together, in the same direction, the process is called *symport*; when the two chemicals pass only with the help of each other, but in opposite directions, one says that we have *antiport* – see, e.g., Reference.[1)]

A mathematical counterpart of these notions is to consider rules of the form $(ab, in), (ab, out)$ (symport), and $(a, in; b, out)$ (antiport). How much can we compute by making use of such rules (and no other type of rules)? At the first sight, not too much, and this makes highly surprising the main result of our paper: P systems with symport and antiport rules are computationally univer-

sal. The proof uses a system with five membranes in order to simulate a Turing machine (actually, we simulate matrix grammars with appearance checking, devices known to be equivalent with Turing machines). The number of membranes can be reduced to two if more complex rules are allowed, for instance, of the form $(ab, in; cd, out)$.

The proofs use both symport and antiport rules. What about using only symport rules? We do not have an answer to this question, but we believe that when the use of rules is allowed only in the presence of certain "promoters" or "inhibitors" we can again compute at a significant level.

§2 P Systems with Symport/Antiport Rules

The language theory notions we use here are standard, and can be found in many monographs, for instance, in Reference.[5] A membrane structure can be represented by a string of matching parentheses. A multiset over a set X is a mapping $M : X \longrightarrow \mathbf{N} \cup \{\infty\}$ (we allow infinite multiplicity). With these simple prerequisites we are ready now to introduce our variant of P systems.

A *P system* (of degree $m \geq 1$) *with symport/antiport rules* is a construct

$$\Pi = (V, \mu, M_1, \ldots, M_m, M_e, R_1, \ldots, R_m, i_o),$$

where: V is the alphabet of *objects*; μ is a membrane structure with m membranes (injectively labeled by positive integers $1, 2, \ldots, m$; in this paper, the skin membrane has always label 1); $M_1, \ldots, M_m$ are the multisets of objects initially present in the regions of the system, and M_e is the multiset of objects present outside the system, in the *environment*; the "internal multisets" have finite multiplicities of objects (hence, we can represent them by strings of objects – the number of occurrences of each object in a string gives the multiplicity of this object in the multiset; of course, permutations of the same string represent the same multiset, and the empty multiset corresponds to the empty string, λ), while for each $a \in V$ we have either $M_e(a) = 0$ or $M_e(a) = \infty$ (outside the system, an object is either absent, or present in arbitrarily many copies; thus, M_e is identified by its *support*, the set of objects which appear at least once – hence arbitrarily many times – in the environment); $R_1, \ldots, R_m$ are finite sets of *rules* of one of the following forms:

- $(a, in), (a, out)$, for $a \in V$ (*uniport rules*);
- $(ab, in), (ab, out)$, for $a, b \in V$ (*symport rules*);
- $(a, out; b, in)$, for $a, b \in V$ (*antiport rules*);

$i_o \in \{1, \ldots, m\}$ is an elementary membrane of μ (the output membrane).

The meaning of the rules in $R_i, 1 \leq i \leq m$, is obvious: an object a can enter (exit) the region of membrane i if the rule (a, in) (resp. (a, out)) is in R_i; if (ab, in) or (ab, out) is present in R_i, then a and b together can enter or exit, respectively; finally, if $(a, out; b, in)$ belongs to R_i, then a exits region i and simultaneously b enters it. In order to use such a rule which involves two objects, both these objects must be present in the respective regions. If in R_1 we have rules $(a, in), (ab, in)$ for objects a, b which are available in the environment in infinitely many copies, then the computation will never stop: using such a

rule brings into the system arbitrarily many copies of a and b, but because we assume that the environment is inexhaustible, the rule can be used again.

The multisets of objects present in the m regions of Π constitute the *configuration* of the system. We pass from a configuration to another configuration by using the rules from $R_1, \ldots, R_m$, as customary in P systems: the rules are applied in the non-deterministic maximally parallel manner, in the sense that we apply the rules in parallel, to all objects which can be processed, non-deterministically choosing the rules and the objects. Thus, a transition means a redistribution of objects among regions (and environment), which is maximal for the chosen set of rules. A sequence of transitions constitutes a *computation*; a computation is successful if it *halts*, i.e., it reaches a configuration where no rule can be applied. The *result* of a successful computation is the number of objects present in the membrane with label i_o in the halting configuration. A computation which never halts yields no result. The set of numbers computed by Π is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$, computed by systems Π of degree at most $m \geq 1$ is denoted by NPP_m. Also, we use NRE to denote the family of recursively enumerable sets of natural numbers. (If we distinguish among the objects present in the output membrane, then we can compute vectors of natural numbers, hence relations, as done in many papers about P systems. The extension is obvious, so we do not enter here into details.)

§3 The Computational Power

We will prove that P systems with the communication done by symport and antiport rules (even with a small number of membranes) are able to simulate Turing machines. In proofs we need the notion of a *matrix grammar with appearance checking*, a class of controlled context-free Chomsky grammars much investigated in formal language theory, see References.[2,5] Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M. For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – therefore we say that these rules are applied in the *appearance checking* mode.)

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. It is known that matrix grammars with appearance checking generate precisely the family RE of recursively enumerable languages.

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms: (i) $(S \to XA)$, with $X \in N_1, A \in$

N_2, (ii) $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$, (iii) $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$, and (iv) $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$. There is only one matrix of type (i) and F consists exactly of all rules $A \to \#$ appearing in matrices of type (iii); $\#$ is a trap-symbol, once introduced, it is never removed. A matrix of type (iv) is used only once, in the last step of a derivation.

It is known that for each matrix grammar there is an equivalent matrix grammar in the binary normal form – see Lemma 1.3.7 from Reference.[2)]

Theorem 3.1
$NPP_m = NRE$, for all $m \geq 5$.

Proof
The inclusions $NPP_m \subseteq NPP_r, r \geq m \geq 1$, are obvious, and $NPP_m \subseteq NRE, m \geq 1$, can be proved in a straightforward manner, so we have only to prove the inclusion $NRE \subseteq NPP_5$. To this aim, we use the equality $RE = MAT_{ac}$, with the obvious consequence that any set from NRE is the length set of a language from MAT_{ac}. In particular, we can consider a language over the one-letter alphabet. Therefore, we can start from a matrix grammar $G = (N, T, S, M, F)$ with $T = \{a\}$, in the binary normal form. Without any loss of the generality, we may assume that there is only one terminal matrix, of the form $(Z \to \lambda, B \to \lambda)$, for fixed $Z \in N_1, B \in N_2$ (we replace each terminal matrix $(X \to \lambda, A \to aa)$ of G by $(X \to X', A \to aA'), (X' \to Z, A' \to aB)$, each matrix $(X \to \lambda, A \to a)$ by $(X \to Z, A \to aB)$, and each matrix $(X \to \lambda, A \to \lambda)$ by $(X \to Z, A \to B)$, and then add the unique terminal matrix $(Z \to \lambda, B \to \lambda)$).

Assume that M contains the matrices $m_i : (X \to Y, A \to \alpha\beta), X, Y \in N_1, A \in N_2, \alpha, \beta \in N_2 \cup \{a, \lambda\}$, for $i = 1, \ldots, k$, and $m_i : (X \to Y, A \to \#)$, $X, Y \in N_1, A \in N_2$, for $i = k+1, \ldots, n$, for some $k \geq 1, n \geq k$. Moreover, denote the initial matrix $(S \to XA)$ of G by $(S \to X_0A_0)$ and label by m_{n+1} the terminal matrix $(Z \to \lambda, B \to \lambda)$.

We construct the system $\Pi = (V, \mu, M_1, \ldots, M_5, M_e, R_1, \ldots, R_5, 5)$, with

$$
\begin{aligned}
V = \;& N_1 \cup N_2 \cup \{a, c, f, t_e, t_2, t_3\} \cup \{d_i, d_i', d_i'', d_i''' \mid 1 \leq i \leq n+1\},\\
\mu = \;& [_1[_2[_3[_4\,]_4]_3]_2[_5\,]_5]_1,\\
M_1 = \;& X_0A_0, \quad M_2 = t_2, \quad M_3 = c^4t_3,\\
M_4 = \;& d_1d_1'd_1''d_1''' \ldots d_{n+1}d_{n+1}'d_{n+1}''d_{n+1}''', \quad M_5 = \lambda,\\
M_e = \;& N_1 \cup N_2 \cup \{a, f, t_e\},\\
R_1 = \;& \{(d_iX, out),\ (d_i'A, out),\ (d_i''d_i''', out),\\
& (d_i, out; t_e, in),\ (d_i', out; t_e, in),\ (d_i'', out; t_e, in),\ (d_i''', out; t_e, in),\\
& (d_iY, in),\ (d_i', in),\ (d_i''\alpha, in),\ (d_i'''\beta, in) \mid 1 \leq i \leq k,\\
& m_i : (X \to Y, A \to \alpha\beta), X, Y \in N_2, \alpha, \beta \in N_2 \cup \{a, \lambda\}, 1 \leq i \leq k\}\\
\cup \;& \{(d_{n+1}Z, out),\ (d_{n+1}'B, out),\ (d_{n+1}'', out; f, in),\ (d_{n+1}''', out),\\
& (d_{n+1}, out; t_e, in),\ (d_{n+1}', out; t_e, in)\}\\
\cup \;& \{(d_iX, out),\ (d_i''d_i''', out),\ (d_iY, in),\ (d_i''d_i''', in),\\
& (d_i, out; t_e, in),\ (d_i'', out; t_e, in),\ (d_i''', out; t_e, in) \mid \text{for}\\
& m_i : (X \to Y, A \to \#), X, Y \in N_1, A \in N_2, k+1 \leq i \leq n\},
\end{aligned}
$$

$$
\begin{aligned}
R_2 = {} & \{(d_i d_i'', out),\ (d_i' d_i''', out),\\
& (d_i t_2, out),\ (d_i' t_2, out),\ (d_i'' t_2, out),\ (d_i''' t_2, out) \mid 1 \le i \le n+1\}\\
\cup {} & \{(d_i' d_i''', in),\ (d_i d_i'', in) \mid 1 \le i \le n\}\\
\cup {} & \{(d_i' A, in) \mid m_i : (X \to Y, A \to \#), k+1 \le i \le n\}\\
\cup {} & \{(fA, in),\ (At_2, out) \mid A \in N_2\} \cup \{(t_3, out)\},\\
R_3 = {} & \{(d_i d_i', out),\ (d_i'' d_i''', out),\\
& (d_i t_3, out),\ (d_i' t_3, out),\ (d_i'' t_3, out),\ (d_i''' t_3, out) \mid 1 \le i \le n+1\}\\
\cup {} & \{(d_i' d_i''', in),\ (d_i d_i'', in) \mid 1 \le i \le n\},\\
R_4 = {} & \{(d_i, out; c, in),\ (d_i', out; c, in),\ (d_i'', out; c, in),\ (d_i''', out; c, in) \mid\\
& 1 \le i \le n+1\}\\
\cup {} & \{(c, out; d_i, in),\ (c, out; d_i', in),\ (c, out; d_i'', in),\ (c, out; d_i''', in) \mid\\
& 1 \le i \le n\},\\
R_5 = {} & \{(a, in),\ (t_2, in),\ (t_2, out),\ (t_3, in),\ (t_3, out),\ (t_e, in),\ (t_e, out)\}.
\end{aligned}
$$

Before examining in some detail the work of this system, let us point out some general facts. The symbols t_2, t_3, t_e are trap-symbols, once arriving in the skin membrane, they will pass forever through membrane 5, hence the computation will never halt (note that the subscripts of these symbols indicate their place in the initial configuration of the system).

Then, with each matrix m_i of G we have associated a four-tuple of symbols, $d_i, d_i', d_i'', d_i''', 1 \le i \le n+1$ (let us call them "controllers"), which will control the simulation of the matrix m_i by a series of transitions among configurations of Π. These symbols are released from membrane 4 by means of the four copies of the symbol c (which can be considered a "trigger"). Crucial for the success of the simulation is the fact that, *if from membrane 4 we release symbols d_i, d_j', d_h'', d_l''' with at least two of the subscripts i, j, h, l different, then one of the trap-symbols will arrive to the skin membrane*, hence the computation will never stop (the trap-symbols will pass forever back and forth through membrane 5). This assertion is easy to check: we can exit membrane 3 only with two pairs $d_i d_i'$ and $d_h'' d_h'''$, hence in this way we check the equalities $i = j$ and $h = l$; then, we can exit membrane 2 only with the pairs $d_i d_i''$ and $d_j' d_j'''$, hence in way we also check the equalities $i = h$ and $j = l$. If no such pair can be formed, or only one pair can be formed, then the remaining symbols have to use the rules of the forms $(d_i t_r, out)$, $(d_j' t_r, out)$, $(d_h'' t_r, out)$, $(d_l''' t_r, out)$, with $r = 2, 3$, for the corresponding membranes r, and in this way t_3 and/or t_2 will be released.

At each moment when the four "controllers" d_i, d_i', d_i'', d_i''' exit a membrane 4, 3, 2, they can return back, but this changes nothing, hence the computation must continue. Moreover, as long as copies of c or of d_i, d_i', d_i'', d_i''' are outside membrane 4, the computation must continue.

Now, the simulation of matrices from M develops differently for (nonterminal) matrices without appearance checking rules, for matrices with appearance checking rules, and for the terminal matrix. After simulating matrices of the first two types, the symbols d_i, d_i', d_i'', d_i''' can return to membrane 3, can release the "triggers" c from membrane 4, and hence the process can be continued by simulating another matrix. However, after simulating the terminal matrix, the "controllers" are left in the environment, and the computation will stop – providing that no nonterminal is present in the system, otherwise the

"checker" f, just introduced when simulating the matrix m_{n+1}, will bring a symbol $A \in N_2$ to membrane 2, and in this way the trap-symbol t_2 is released, by the rule $(At_2, out) \in R_2$.

Another important general observation is that the simulation of matrices $m_i, 1 \leq i \leq n$, should be complete, in the sense that both their rules should be simulated, otherwise the trap-symbols are introduced; this is ensured by the rules which exchange symbols d_i, d_i', d_i'', d_i''' with t_e, which are present in R_1 for all $1 \leq i \leq n$ – except that for $k+1 \leq i \leq n$ we do not have the rule $(d_i', out; t_e, in)$ in R_1, because d_i' has the task to check the presence of a specific symbol in the skin region, as we will immediately see.

In any moment, any copy of a is sent to the output membrane (hence from now on we will ignore this symbol).

Consider now a configuration where the symbols d_i, d_i', d_i'', d_i''', for some $1 \leq i \leq k$ such that $m_i : (X \to Y, A \to \alpha\beta)$, have arrived in the skin membrane. By using the rules $(d_iX, out), (d_i'A, out), (d_i''d_i''', out)$ and then $(d_iY, in), (d_i', in)$, $(d_i''\alpha, in), (d_i'''\beta, in)$ from R_1, we correctly simulate the matrix m_i (note that each of α, β can be empty). If, say, only the rule (d_iX, out) is used, but d_i' returns to membrane 2 together with d_i''' (by means of the rule $(d_i'd_i''', in) \in R_2$), then d_i'' will have to use the rule $(d_i'', out; t_e, in) \in R_1$, and the trap-symbol t_e gets in. The same result is obtained when we simulate only the second rule of the matrix m_i.

After simulating the matrix m_i, the symbols d_i, d_i', d_i'', d_i''' can simulate again the same matrix (if $X = Y$), and this is correct with respect to G, or they can return to membrane 3, and from here to membrane 4, releasing the four copies of the symbol c, which have waited in membrane 4 during the simulation.

When we start by a four-tuple d_i, d_i', d_i'', d_i''' with $k+1 \leq i \leq n$, hence corresponding to a matrix $m_i : (X \to Y, A \to \#)$, then again d_i exits the system (by the rule $(d_iX, out) \in R_1$), but, if any copy of A is present in membrane 1, then the symbol d_i' has to use the rule $(d_i'A, in) \in R_2$; in this way, A gets into membrane 2 and will immediately exit together with the trap-symbol t_2. If no copy of A is present in the skin membrane, then d_i' waits. In the next step, d_i'', d_i''' will return from the environment, hence all four symbols d_i, d_i', d_i'', d_i''' can return to membrane 4 (they can simulate again the matrix m_i only if $X = Y$, but this changes nothing).

After simulating a matrix m_i with $1 \leq i \leq n$, we return to a configuration as that we have started with, with the auxiliary symbols "stored" in the central membranes and all symbols from $N_1 \cup N_2$ in the skin membrane, hence the process can be iterated.

When simulating the terminal matrix $m_{n+1} : (Z \to \lambda, B \to \lambda)$, the four symbols $d_{n+1}, d_{n+1}', d_{n+1}'', d_{n+1}'''$ are brought to the skin membrane as above, d_{n+1} and d_{n+1}' check whether or not Z and B are present (otherwise the trap-symbol t_e is introduced), d_{n+1}'' is exchanged for f (by means of the rule $(d_{n+1}'', out; f, in) \in R_1$), and d_{n+1}''' just exits the system. If any symbol from N_2 is still present in the skin membrane, then it will go together with f to membrane 2, and will return together with t_2; if no symbol from N_2 is present, that is, the derivation in G is terminal, then f remains unused, and the computation stops (none of the symbols $d_{n+1}, d_{n+1}', d_{n+1}'', d_{n+1}'''$ is brought back from the environment, hence

there is no way to release again the "trigger" c from membrane 4).

Having in mind the explanations from the beginning of this discussion about the work of the system Π, the reader can easily check other possible branches of the computations in Π, always finding that the only way to get a halting computation is to correctly simulate derivations in G. Consequently, $N(\Pi) = \{m \mid a^m \in L(G)\}$, and this concludes the proof. ■

§4 More Complex Rules

In the previous section we have tried to be "realistic," using rules of the forms encountered in biology, with at most two chemicals collaborating in their passage through membranes. From a mathematical point of view it is natural to consider rules of a general form, $(u, out; v, in)$, with u and v strings of an arbitrary length. The pair of numbers $(|u|, |v|)$ is called the *radius* of the rule. In the previous section we have used only rules of the radius (1, 0), (2, 0), (0, 1), (0, 2), (1, 1). As it is expected, when using rules of a larger radius, characterizations of NRE can be obtained by P systems with a number of membranes smaller than in Theorem 3.1. Actually, two membranes suffice, and the price for this decrease in the number of membranes is not too big: rules of radius at most (componentwise) (2, 2) suffice.

Let us denote by $NPP_m(r, s)$ the family of sets of natural numbers computed by P systems with at most $m \geq 1$ membranes, using rules of radius componentwise at most (r, s). When one of m, r, s is not bounded, we replace it with $*$.

With the present terminology, P systems with carriers can be considered as systems with symport and antiport communication, and the main result from Reference[3] says that rules of radius at most $(3, 3)$ suffice. The following result is stronger from this point of view.

Theorem 4.1
$NPP_m(r, s) = NRE$ for all $m \geq 2$ and (r, s) componentwise larger than or equal to $(2, 2)$.

Proof
Again, we only have to prove the inclusion $NRE \subseteq NPP_2(2, 2)$. Let $G = (N, \{a\}, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form, and containing a unique terminal matrix, $(Z \to \lambda, B \to \lambda)$. With the same notations as in the proof of Theorem 3.1, we construct the P system $\Pi = (V, [_1[_2\,]_2]_1, M_1, M_2, M_e, R_1, R_2, 2)$, with

$$
\begin{aligned}
V = \;& N_1 \cup N_2 \cup \{a, c, f, g, h, t\} \cup \{c_i, c_i', d_i \mid 1 \leq i \leq n+1\},\\
M_1 = \;& cX_0A_0, \quad M_2 = \lambda,\\
M_e = \;& N_1 \cup N_2 \cup \{a, f, g, h, t\} \cup \{c_i, c_i', d_i \mid 1 \leq i \leq n\},\\
R_1 = \;& \{(cX, out; c_iY', in),\ (c_iA, out; cc_i', in),\ (c_i', out; u, in),\ (c_i, out; t, in) \mid\\
& m_i : (X \to Y, A \to u), 1 \leq i \leq k\}\\
\cup \;& \{(c, out; t, in),\ (cZ, out; c_{n+1}f, in),\ (c_{n+1}A, out),\ (c_{n+1}, out; t, in)\}
\end{aligned}
$$

$$\begin{aligned} &\cup \{(cX, out; c_i d_i, in),\ (d_i, out; Yh, in),\ (c_i A, out; t, in),\\ &\quad (h, out; cg, in),\ (c_i g, out) \mid m_i : (X \to Y, A \to \#), k+1 \le i \le n\}\\ &\cup \{(fD, out; t, in) \mid D \in N_2\},\\ R_2 = &\{(a, in),\ (t, in),\ (t, out)\}. \end{aligned}$$

The symbols c, c_i, c_i' control the simulation of matrices $m_i, 1 \le i \le k$, in the following way. After using the rule $(cX, out; c_i Y, in) \in R_1$, we have c_i in the system. If the rule $(c_i A, out; c_i' c, in) \in R_1$ cannot be used, then the symbol c_i will bring the trap-symbol t into the system, and this symbol will go forever back and forth through membrane 2, hence the computation will never finish. If the rule $(c_i A, out; c_i' c, in) \in R_1$ is used, then at the next step c_i' will exit, bringing into the system the string u, which completes the simulation of the matrix $m_i : (X \to Y, A \to u)$.

In the case of the terminal matrix, we bring the symbols c_{n+1}, f in the system, c_{n+1} simulates the second rule of the matrix, and f checks whether or not the derivation was a terminal one. In the negative case, the rules $(fD, out; t, in) \in R_1$, for $D \in N_2$, will bring the trap-symbol in the system.

If we start with a rule of the form $(cX, out; c_i d_i, in) \in R_1$, for some m_i : $(X \to Y, A \to \#), k+1 \le i \le n$, then at the next step we have to use the rule $(d_i, out; Yh, in) \in R_1$. If in the skin membrane there is any copy of A, then the rule $(c_i A, out; t, in)$ has to be used, and the computation will never finish, because of the rules $(t, in), (t, out)$ from R_2. If no copy of A is present, then c_i waits in the skin membrane. At the next step, h exits and brings cg inside. Now, together with g, the symbol c_i can leave the system. In this way, the application of matrix m_i was simulated.

In all cases the symbol c is again available, hence the process can be iterated. Consequently, $N(\Pi) = \{m \mid a^m \in L(G)\}$. (It is worth noting that the main part of the work of our system is done by the interaction between the skin membrane and the environment, membrane 2 is used only for storing the result of the computation and in order to prevent the "wrong" computations, by using forever the trap-symbol after introducing it into the system.) ■

As it is also the case for Theorem 3.1, we do not know whether this result is optimal (this time, we have to consider not only the number of membranes, but also the radius of the rules), but systems of simpler forms seem not to be too powerful. The easy proof of the next result is left to the reader.

Theorem 4.2
Families $NPP_1(*, 1), NPP_*(*, 0), NPP_*(0, *)$ contain only finite sets of numbers, but families $NPP_1(1, 2)$ and $NPP_2(1, 1)$ contain infinite sets of numbers.

§5 Final Remarks

Starting from the biological observation that the membrane transport is often performed by pairs of chemicals which help each other in this process (called in biology symport or antiport), we have considered a purely communicative variant of P systems, where no object is changed during the computation, but

only the places of objects are changed. This means that the conservation law is observed in this framework (which is not necessarily the case for most of the previously considered classes of P systems). Surprisingly enough, this kind of "osmotic computation" proves to be universal, all Turing computable sets of numbers can be computed in this way. Moreover, this is done by a class of P systems with the rules strictly corresponding to the case from biology, where at most two chemical compounds are collaborating in passing through membranes.

Five membranes were used in our proof; the number of membranes can be decreased to two if more complex (antiport) rules are allowed, that is, with more than two objects collaborating in their passage through membranes. This indicates an expected trade-off between the number of membranes and the complexity (the radius) of symport/antiport rules – this trade-off deserves to be further investigated.

From a mathematical point of view, it also remains to be investigated the case when only symport rules are used, that is, only rules $(u, in), (u, out)$, for u consisting of at most two symbols. Because we feel that such rules are not very powerful, a possible way to increase their power is to apply them conditionally, under the control of *permitting* or *forbidding* symbols. Specifically, a rule $(u, in)_b$, or $(u, out)_b$ associated with a region i can be used only if b is present in this membrane. This is somewhat similar to the case of antiport rules, as b supports the exit of u, but not leaving the region at the same time with u. Similarly, we can consider forbidding symbols, in the presence of which a rule cannot be used. This corresponds to the promoters and inhibitors known from biology, hence it is again a "realistic" variant. How powerful it is, it remains to be investigated. We expect that the family of sets of numbers computed by P systems which use only symport rules applied in a conditional manner is significantly large. For instance, using only permitting conditions we can already compute non-semilinear sets of numbers. We will return to this topic in a forthcoming paper.

Acknowledgements

Many thanks are due to two anonymous referees, for a very careful reading of a previous version of this paper.

References

1) Alberts, B., et al., *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.

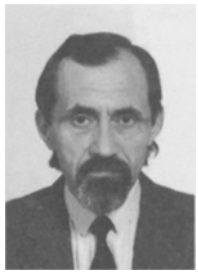
2) Dassow, J. and Păun, Gh., *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

3) Martin-Vide, C., Păun, Gh. and Rozenberg, G., "Membrane Systems with Carriers," *Theoretical Computer Science, 270*, pp. 779–796, 2002.

4) Păun, Gh., "Computing with Membranes," *Journal of Computer and System Sciences, 61, 1*, pp. 108–143, 2000.

5) *Handbook of Formal Languages* (Rozenberg, G. and Salomaa, A., eds.), Springer-Verlag, Berlin, 1997.

Andrei Păun: He graduated the Faculty of Mathematics of Bucharest University in 1998, received his M.Sc. degree from The University of Western Ontario in 1999, and since then he is a PhD student in the Computer Science Department of University of Western Ontario, London, Canada (under the guidance of prof. Sheng Yu). The topic of his thesis is Molecular Computing (especially, DNA and Membrane Computing), but his research interests also include neural networks, implementing automata, combinatorics on words.



Gheorghe Păun: (the proud father of two sons, including the first author of this paper) He is a member of the Romanian Academy, working as a senior researcher in the Institute of Mathematics of the Romanian Academy, Bucharest, and as a Ramon y Cajal researcher in Rovira i Virgili University of Tarragona, Spain. He is one of the most active authors in (the theory of) DNA Computing, (co)author of many papers in this area, (co)author and (co)editor of several books. In 1998 he has initiated the area of Membrane Computing. Other research interests: regulated rewriting, grammar systems, contextual grammars, combinatorics on words, computational linguistics.