View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by idUS. Depósito de Investigación Universidad de Sevilla

Bridging Membrane and Reaction Systems – Further Results and Research Topics

Gheorghe Păun[1,2], Mario J. Pérez-Jiménez[2], Grzegorz Rozenberg[3]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
[2] Research Group on Natural Computing
Department of Computer Science and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es
[3] Leiden Institute for Advanced Computer Science - LIACS
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Summary. This paper continues an investigation into bridging two research areas concerned with natural computing: membrane computing and reaction systems. More specifically, the paper considers a transfer of two assumptions/axioms of reaction systems, non-permanency and the threshold assumption, into the framework of membrane computing. It is proved that: (1) SN P systems with non-permanency of spikes assumption characterize the semilinear sets of numbers, and (2) symport/antiport P systems with threshold assumption (translated as ω multiplicity of objects) can solve SAT in polynomial time. Also, several open research problems are stated.

1 Introduction

This paper continues research aimed at bridging two research areas concerned with processes inspired by the functioning of living cells, *membrane computing* (see, e.g., [10], [11], [14]) and reaction systems (see, e.g., [1], [3] – [6]). Membrane computing (based on *P systems*) essentially deals with *multisets*, processed in the compartments of a *membrane structure* according to rules of various types, such as, e.g., multiset rewriting and symport/antiport rules. Thus, the objects are present with specified multiplicity within the regions delimited by membranes, some of them evolve by the rules associated with membranes while the objects which are not involved in the rules used at a given step remain unchanged – thus they can be used in the subsequent processing steps.

The situation is very different in reaction systems. First of all, because of the assumed abstraction level this is a qualitative model, i.e., there is no counting: one

deals with sets rather than with multisets. Consequently, it is assumed that if an entity is present, then it is present in enough copies to be used by all reactions that use this entity as a reactant. This is referred to as a *threshold assumption*. Secondly, an entity is present in a successor state T' of a given state T only if it is produced by a reaction enabled in T or it is put into T' by the environment/context. This reflects the basic bioenergetics of the living cell, and it is referred to as the *non-permanency assumption*.

In this paper we continue an investigation of bridging membrane computing and reaction systems (see [12] and [13]) by transferring the threshold and the non-permanency assumptions to the framework of P systems. In particular, we investigate the resulting computing power and efficiency of some classes of P systems. We prove that:

(1) spiking neural (in short, SN) P systems with non-permanency of spikes characterize/compute just semilinear sets of numbers, while traditional SN P systems are Turing complete,

(2) symport/antiport P systems with the threshold assumption can solve **NP**-complete problems in polynomial time – this is illustrated with SAT, the satisfiability of propositional formulas in the conjunctive normal form.

We conclude this paper by stating a number of research problems.

2 Prerequisites

We assume the reader to be familiar with basic elements of membrane computing (e.g., from [11], [14], [17]) and of language theory (e.g., from [16]). Here we only recall some general notions and notations.

The language of all strings over an alphabet V is denoted by V^*, the empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$.

We denote by $SLIN_1$ the family of semilinear sets of numbers, and by NRE the family of recursively enumerable (Turing computable) sets of numbers. Semilinear sets are the length sets of regular languages, which are languages characterized by *regular expressions* or generated by *regular grammars*. A regular grammar is specified in the form $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom of the grammar, and P is a set of rules, each of which is of the form $A \to aB$, $A \to a$, where $A, B \in N, a \in T$.

As customary in membrane computing, we represent the multisets over an alphabet V by strings in V^* (hence a string and all its permutations represent the same multiset). Thus, we speak interchangeably about strings and multisets over V, and $|w|$ represents both the length of the string w and the cardinality of the multiset (represented by) w. We also write $w \subseteq w'$ for the inclusion between multisets (represented by the strings) w and w'. Since a set M is a multiset where all elements have multiplicity one, it is represented by a string containing each symbol from M exactly once.

2.1 Membrane Computing

We briefly recall here two classes of P systems which we investigate in this paper: the *symport/antiport P systems*, [9], and the *spiking neural* (SN) *P systems*, [8].

A membrane structure is a cell-like hierarchical arrangement of labeled membranes (understood as 3D vesicles); the external membrane is usually called the *skin* membrane, and a membrane without any membrane inside is called *elementary*. With each membrane, one associates a *region*, which is the space delimited by it and the inner membranes, if any. A membrane structure can be naturally represented by a rooted tree or by an expression of labeled parentheses (with a unique external parenthesis, associated with the skin).

A *symport* rule is either of the form (x, in) or of the form (x, out), and an *antiport* rule is of the form $(z, out; w, in)$, where x, z, and w are multisets of objects. These rules formalize the biological operations of moving several objects at a time across a membrane, either in the same direction, as is the case for symport rules or in opposite directions, as is the case for antiport rules.

A *P system with symport/antiport rules* is a construct of the form

$$\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_{in}, i_{out}),$$

where O is an alphabet of objects, μ is a membrane structure with m membranes (here, labeled by $1, \ldots, m$, but any set of labels associated in a one-to-one manner to membranes can be used), $w_1, \ldots, w_m$ are the multisets present in the initial configuration in the m regions of μ (delimited by membranes labeled by $1, \ldots, m$, respectively), $E \subseteq O$, $R_1, \ldots, R_m$ are finite sets of symport/antiport rules associated with the m membranes of μ, and i_{in}, i_{out} are the input and the output regions of the system (i_{in} indicates a region of μ, while i_{out} can also be the environment of the system – we write then $i_{out} = env$). The objects of E are supposed to be present in the environment of the system with an arbitrary multiplicity.

Using an antiport rule $(z, out; w, in)$ associated with a membrane i means sending the multiset z out of region i and, simultaneously, bringing the multiset w into membrane i from the outside region adjacent to membrane i. Similarly for symport rules, where only one multiset of objects is moved across membrane i.

(Note that the symport/antiport rules do not change the number of objects, but they only displace them – that is why we need a supply of objects in the environment; this supply is inexhaustible, i.e., it does not matter how many objects are introduced into the system, still arbitrarily many objects remain in the environment.)

The rules are used in the nondeterministic maximally parallel manner. In the initial configuration, an *input* is introduced into region i_{in} in the form of a multiset, and the *result* of a computation is given in region i_{out}, most typically at the end of the computation (when no rule can be applied). If the system is used in the generative mode, then i_{in} is ignored/removed. If the system is used in the accepting mode, then i_{out} is ignored and the input is accepted if and only if the computation halts. In the computing mode both i_{in} and i_{out} are used. In particular, the system

can be used in the decidability mode: the code of an instance of a decision problem is introduced in i_{in} and the result is obtained in i_{out}: an object `yes` is placed in i_{out} at a specified step of the computation if and only if the instance of the problem has a positive answer.

More precise definitions of what it means to solve a decision problem in terms of P systems (complexity classes, uniform versus semi-uniform solutions, frontiers of efficiency etc.) can be found in many places – we mention here only [15] and the corresponding chapter from [14]. Several results in this area say that P systems able to produce an exponential workspace in a linear time (e.g., by membrane division, membrane creation, string replication) can solve computationally hard problems (typically, **NP**-problems) in polynomial time; the term *fypercomputation* was proposed in [12] for this situation, a sort of analogy to an established term *hypercomputation*, see, e.g., [2].

It is known that symport/antiport P systems (with a small number of membranes and with rules of a low complexity) used in the generative or the accepting mode characterize NRE (see, e.g., [14]).

Note that in the previous definitions multisets play a crucial role: objects appear with a finite multiplicity and the objects which do not evolve by a rule remain unchanged. However, the objects from the set E are used according to the threshold assumption (but they do not obey the non-permanency assumption).

The threshold assumption can be applied to some or to all membranes of a symport/antiport P system. In such *distinguished* membranes (where the threshold assumption applies), any object – even if it comes from a neighboring membrane with a specified multiplicity, maybe in only one copy – is present in arbitrarily many copies. A P system with such membranes is said to be an ωP *system.*

Another class of P systems investigated in this paper is that of *spiking neural P systems*, in short, SN P systems. Such a system (with extended rules, without delay, of degree $m \geq 1$) is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons* of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   b) R_i is a finite set of *rules* each of which is in one of the following two forms:
      (1) $E/a^c \to a^p$, where E is a regular expression over $\{a\}$ and $1 \leq p \leq c$;
      (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \in L(E)$ for no rule $E/a^c \to a$ of type (1) from R_i;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);

4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a^p$ can be applied, and this means that c spikes are consumed, only $k - c$ remain in the neuron, and p spikes are produced and submitted to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). The rules of type (2) are *forgetting* rules: if the neuron contains exactly s spikes, then the rule $a^s \to \lambda$ can be used, and this means that all s spikes are removed. The rules are used in the sequential manner within each neuron, and in parallel for all neurons of the system.

Using the rules as described above (see more detailed/precise definitions in the literature), we can define transitions among configurations. With a computation we can associate a result in several ways. The basic one associates a number to each computation (halting or not), viz., as the number of steps elapsed between the first and the second time when the output neuron spikes. The set of such numbers "generated" by Π is denoted by $N_2(\Pi)$. Another possibility is to count all spikes sent to the environment by the output neuron during halting computations. The set of numbers computed by Π in this way is denoted by $N_{out}(\Pi)$.

For both modes two types of results were obtained: Turing computability in the case of neurons without any bound on the number of spikes present inside, and a characterization of semilinear sets of numbers in the case of systems whose neurons have a bound on the number of spikes (we also call such systems *bounded*).

Note that also for SN P systems the multisets (counting the spikes in each neuron) and the permanency (spikes unused remain in the neurons) are essential.

3 The Effect of Non-Permanency

The non-permanency feature was considered in [13] for two classes of P systems: cooperative transition P systems and symport/antiport P systems, and in both cases the universality was proved. Hence there is no loss of power with respect to the traditional membrane computing case, where the objects which do not evolve survive. The case of catalytic P systems was stated as an open problem – recall that under the permanence assumption catalytic P systems are universal, even with two catalysts only, [7]. The effect of non-permanency was not investigated neither for non-cooperative transition P systems nor for the spiking neural P systems.

Let us denote by $N_{out}SNP_m(np)$ the family of sets of numbers $N_{out}(\Pi)$, for SN P systems Π with at most m neurons having the non-permanency property. When the generated numbers are taken as the number of steps between the first two steps when spikes are emitted by the output neuron, then we replace the subscript *out* by 2. The subscript m is replaced by $*$ if no bound on the number of neurons is assumed.

Lemma 1. $N_\alpha SNP_*(np) \subseteq SLIN_1$, *for* $\alpha \in \{2, out\}$.

Proof. If an SN P system Π (with m neurons) works in the non-permanency mode, then after each computation step each neuron has a number of spikes which is bounded by a constant depending on Π: each neuron emits a number of spikes, bounded by the maximum number of spikes produced by any rule of the system – denote this number by M. The spikes produced by a neuron σ_i can be replicated and submitted to at most $m-1$ other neurons (to which there is a synapse from σ_i), hence in total we have at most $m(m-1)M$ spikes. We start with a given initial number of spikes and at any moment we have in the system a bounded number of spikes, distributed to m neurons, which means a finite number of possible configurations of the system. These configurations can be taken as states of a finite automaton (or the nonterminals of a regular grammar) which simulates the work of the system. Taking as a result of a computation (of the automaton or of the grammar) either all spikes sent to the environment by the output neuron of Π or the number of steps between the first two spikes sent to the environment (e.g., we record in the configuration-nonterminal the fact that a spike was emitted, then we "count" until a second spike is emitted, and in that moment we stop the computation of the grammar), we obtain the inclusions of the lemma.

The above reasoning actually shows that an SN P system with non-permanency cannot use rules of the form $E/a^c \to a^p$ where for the regular expression E its language $L(E)$ is infinite. Thus, we can assume that each neuron contains only bounded rules, which then implies the semilinearity of the generated set of numbers (see already [8]).

Also the converse of the previous lemma holds. It was proved in [8] for bounded SN P systems, but the proof in [8] is rather complex (it starts from the characterization of semilinear sets of numbers as the union of a finite set with a finite number of arithmetical progressions), and it does not provide a bound on the number of neurons. Here we provide a direct proof (also bounding the number of neurons), starting from the characterization of semilinear sets of numbers as the length sets of regular languages. Of course, it is sufficient to consider regular languages over the one-letter alphabet.

Lemma 2. $SLIN_1 \subseteq N_{out}SNP_5(np)$.

Proof. Let us consider a regular grammar $G = (N, \{a\}, S, P)$ and assume that $N = \{A_1 = S, A_2, A_3, \ldots, A_n\}$. We construct the following SN P system (its initial configuration is given in a graphical form in Figure 1):

$$
\begin{aligned}
&\Pi = (\{a\}, \sigma_1, \ldots, \sigma_5, syn, 5), \text{ where}\\
&\sigma_1 = (n+1, \{a^{n+i} \to a^n \mid 1 \le i \le n\}),\\
&\sigma_2 = (0, \{a^{n+i} \to a^n \mid 1 \le i \le n\}),\\
&\sigma_3 = (n+1, \{a^{n+i} \to a^j \mid 1 \le i, j \le n, A_i \to aA_j \in P\}\\
&\quad \cup \{a^{n+i} \to a^{n+i} \mid 1 \le i \le n, A_i \to a \in P\}),\\
&\sigma_4 = (0, \{a^{n+i} \to a^j \mid 1 \le i, j \le n, A_i \to aA_j \in P\}
\end{aligned}
$$

$$\cup \{a^{n+i} \to a^{n+i} \mid 1 \le i \le n, A_i \to a \in P\}),$$
$$\sigma_5 = (0, \{a^i \to a \mid 1 \le i \le 2n\}),$$
$$syn = \{(1,2),(2,1),(1,4),(4,1),(2,3),(3,2),(3,4),(4,3),(3,5),(4,5)\}.$$

With each nonterminal $A_i, 1 \le i \le n$, we associate $n+i$ spikes, in neurons σ_3 and σ_4, where the rules in P are simulated. Initially, only neurons σ_1 and σ_3 spike, sending spikes to "partner neurons" σ_2 and σ_4; when these later neurons spike, they send spikes to the former neurons. The computation consists of such alternating steps. Neurons σ_3 and σ_4 also send spikes to the output neuron, σ_5, which sends a spike to the environment in each step. When a terminal rule $A_i \to a$ in P is simulated, neurons σ_3 and σ_4 produce $n+i$ spikes. The output neuron spikes once again, but all other neurons stop working: there is no rule which processes $2n+i$ spikes (these spikes are removed because of the non-permanency axiom, but this is not important since the computation halts anyway).

Consequently, the system Π produces k spikes if and only if $a^k \in L(G)$, hence $SLIN_1 \subseteq N_{out}SNP_5(np)$.

Lemma 3. $SLIN_1 \subseteq N_2SNP_5(np)$.

Proof. We consider the SN P system from the proof of Lemma 2, but now we replace the output neuron σ_5 by the following neuron:

$$\sigma_5 = (a^{2n+1}, \{a^{2n+1} \to a\} \cup \{a^{n+i} \to a \mid 1 \le i \le n\}).$$

The output neuron spikes in the first step and then it spikes only one step after the moment when a rule $A^i \to a$ was simulated in one of the neurons σ_3 or σ_4. In the steps for which $a^i, 1 \le i \le n$, spikes are sent to neuron σ_5; these spikes are removed – this is implied by the non-permanency axiom, because there is no rule to process them.

Consequently, the modified system spikes twice, at a distance of k steps if and only if $a^k \in L(G)$, hence $SLIN_1 \subseteq N_2SNP_5(np)$.

Combining the previous three lemmas we obtain:

Theorem 1. $SLIN_1 = N_\alpha SNP_\beta(np)$, *for all* $\alpha \in \{2, out\}$ *and* $\beta \in \{5, 6, \ldots\} \cup \{*\}$.

What about the SN P systems using less than five neurons? It is easy to see that computations in one-neuron systems last only one step, hence they produce only finite sets. SN P systems with two neurons can generate infinite sets – in the *out* mode. Here is an example of such a system:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \{(1,2),(2,1)\}, 2), \text{ where}$$
$$\sigma_1 = (2, \{a^2 \to a^2,\ a^2 \to a\}),$$
$$\sigma_2 = (0, \{a^2 \to a^2\}).$$

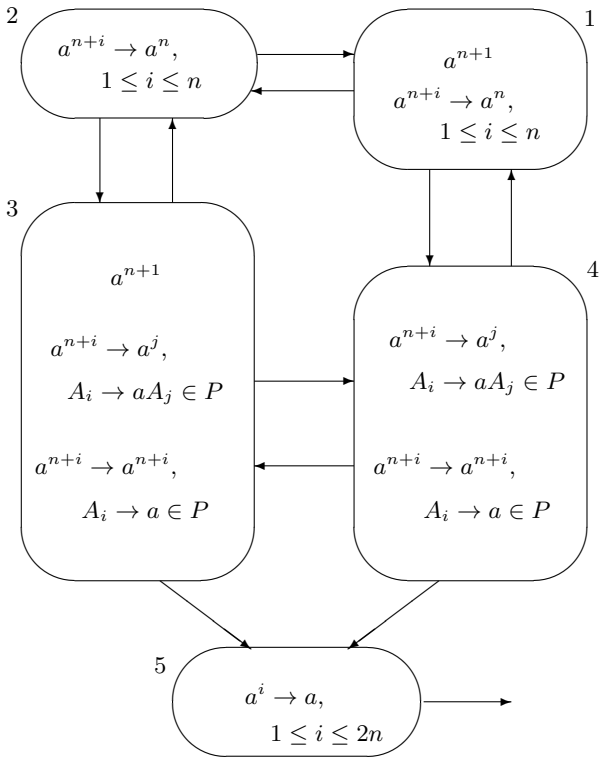


Fig. 1. The SN P system in the proof of Lemma 2

The computation can continue until using the rule $a^2 \to a$ in the first neuron – at that moment no rule can be applied in any neuron and all spikes vanishes. We have $N_{out}(\Pi) = \{2n \mid n \ge 0\}$.

Interestingly enough, when the result is the distance between the first two spikes, SN P systems with two neurons generate only singletons. If there is only one synapse between the two neurons, then each computation lasts one or two steps, hence only one of the numbers 0 and 1 can be generated. If the two neurons can communicate with each other, this can be done simultaneously or at most in alternate steps (after using a rule, no spikes remains in a neuron, because of the non-permanency assumption, hence new spikes must be obtained from the partner neuron); one of the neurons is the output neuron, hence it must spike twice in the first four steps of the computation and so only numbers 1 and 2 can be computed. One can easily see that the generated set is a singleton, containing one of the numbers 1, 2.

However, SN P systems with three neurons can generate infinite sets also as the distance between the first two spikes sent to the environment. This is the case

for the system Π in Figure 2, for which we have $N_2(\Pi) = \{n \mid n \geq 1\}$. The output neuron spikes in the first step of a computation and then only after the step when neuron σ_2 uses the rule $a?2 \to a$; as long as both σ_1 and σ_2 use their rules $a^2 \to a^2$, neuron σ_3 cannot use its rule, hence the four spikes it receives are lost, due to the non-permanency assumption.

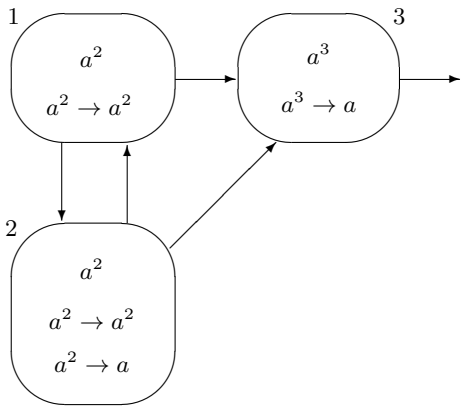


Fig. 2. An SN P system with three neurons generating an infinite set.

It remains an open problem whether SN P systems with four neurons characterize $SLIN_1$.

4 The Effect of the Threshold Assumption

It was proved in [12] that cooperative P systems without permanency, with two membranes where the inner one works under the threshold assumption (any object present here is available in arbitrarily many copies), can solve `SAT` in a polynomial time (actually, linear with respect to the number of clauses and independent of the number of variables) in a uniform way.

This result can be extended to symport/antiport P systems, with one additional feature: the system uses precomputed resources. More specifically, for a $SAT(n, m)$ problem (n variables and m clauses) we work with a number of objects of the order of n^m, i.e., all sets of at most m variables. These objects are given in advance, available in the initial configuration of the system, but they are "precomputed", provided at no cost, although there are exponentially many of them (but without containing other information than that provided by n and m). Of course, in this case we do not work in the standard complexity framework, as the systems solving a class of problems cannot be constructed in a polynomial time with respect to the size of the problems (actually, up to now there is no definition of complexity classes for the case of precomputed resources).

The proof follows the same idea as that of the result in [12], implemented for symport/antiport systems. Since in such systems the objects are only moved across membranes and they cannot be changed during the computation, we see no way to avoid using an exponential number of objects given in advance (in [12] these objects are created during the computation by means of cooperative multiset rewriting rules).

Theorem 2. SAT *can be solved (in a uniform way) in a polynomial time by symport/antiport ωP systems with precomputed resources.*

Proof. Let us consider the SAT problem for n variables, $x_1, x_2, \ldots, x_n$, and m clauses. We denote $Lit = \{x_i, \neg x_i \mid 1 \leq i \leq n\}$ and $Val = \{t_i, f_i \mid 1 \leq i \leq n\}$. Let $v(x_i) = t_i$ and $v(\neg x_i) = f_i$, for all $1 \leq i \leq n$. We have $\neg t_i = f_i$ and $\neg f_i = t_i$, for $1 \leq i \leq n$. (Note that t_i, f_i identify, by their subscripts, the variables with which they are associated.)

An instance $\gamma = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ of $\mathtt{SAT}(n, m)$, with $C_i = y_{i,1} \vee y_{i,2} \vee \ldots \vee y_{i,k_i}$, for $y_{i,j} \in Lit, 1 \leq j \leq k_i$, is encoded as

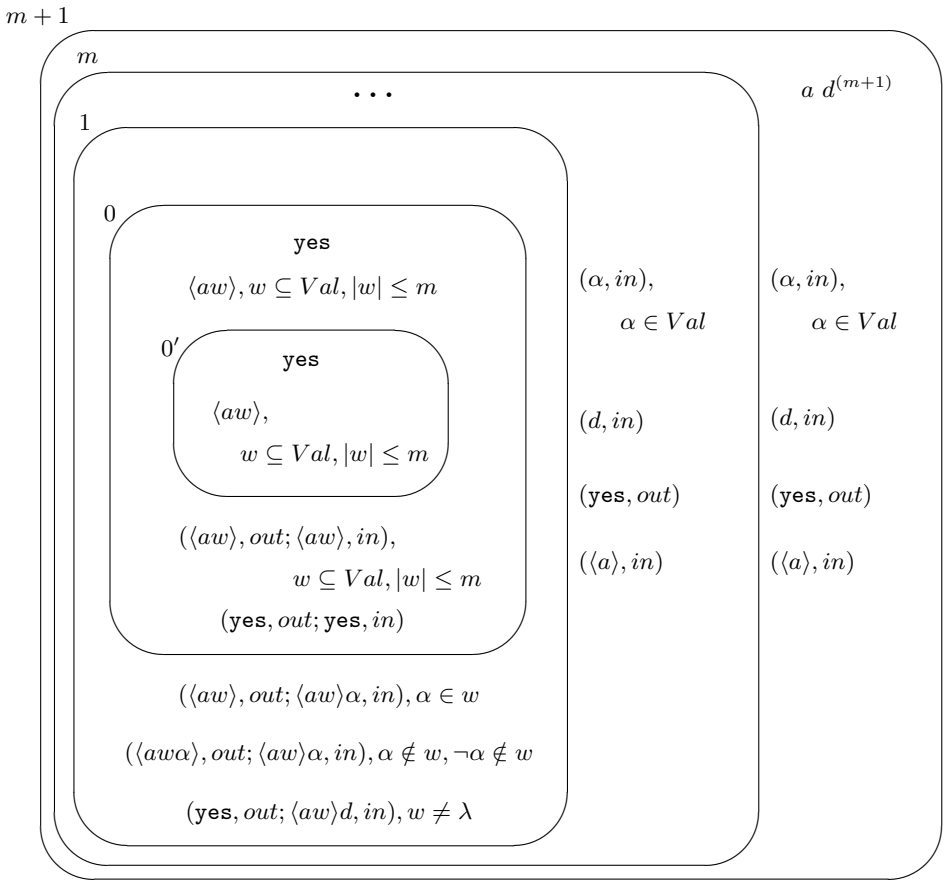
$$code(\gamma) = v(y_{1,1})^{(1)} \ldots v(y_{1,k_1})^{(1)} v(y_{2,1})^{(2)} \ldots v(y_{2,k_2})^{(2)} \ldots v(y_{m,1})^{(m)} \ldots v(y_{m,k_m})^{(m)}.$$

We now construct the following symport/antiport ωP system (the system works under the non-permanency assumption, the three inner membranes are distinguished, and the objects are present in them with ω multiplicity):

$$\begin{aligned}
\Pi &= (O, \mu, w_1, \ldots, w_{m+1}, w_0, w_{0'}, E, R_1, R_2, \ldots, R_{m+1}, R_0, R_{0'}, m+1, env),\\
O &= \{\alpha^{(j)} \mid \alpha \in Val, 1 \leq j \leq m\} \cup Val\\
&\quad \cup \{a, \langle a \rangle, \mathtt{yes}\} \cup \{d^{(i)} \mid 1 \leq i \leq m+1\} \cup \{\langle aw \rangle \mid w \subseteq Val, |w| \leq m\},\\
\mu &= [[\ \ldots [[[[\ ]_{0'}]_0]_1]_2 \cdots]_m]_{m+1},\\
w_{0'} &= w_0 = \{\mathtt{yes}\} \cup \{\langle aw \rangle \mid w \subseteq Val, |w| \leq m\},\\
w_1 &= w_2 = \ldots = w_m = \lambda,\ w_{m+1} = ad^{(m+1)},\\
E &= \{d^{(i)} \mid 1 \leq i \leq m\} \cup \{d, \langle a \rangle\} \cup Val,\\
R_{0'} &= \{(\mathtt{yes}, out; \mathtt{yes}, in\} \cup \{(\langle aw \rangle, out; \langle aw \rangle, in) \mid w \subseteq Val, 0 \leq |w| \leq m\},\\
R_0 &= \{(\langle aw \rangle, out; \langle aw \rangle \alpha, in) \mid \alpha \in Val, w \subseteq Val, 0 \leq |w| \leq m, \alpha \in w\}\\
&\quad \cup \{(\langle aw\alpha \rangle, out; \langle aw \rangle \alpha, in) \mid \alpha \in Val, w \subseteq Val, 0 \leq |w| \leq m, \alpha \notin w, \neg\alpha \notin w\}\\
&\quad \cup \{(\mathtt{yes}, out; \langle aw \rangle d, in) \mid w \subseteq Val, 1 \leq |w| \leq m\},\\
R_i &= \{(\alpha, in) \mid \alpha \in Val\} \cup \{(d, in),\ (\mathtt{yes}, out),\ (\langle a \rangle, in)\},\ 1 \leq i \leq m,\\
R_{m+1} &= \{(d^{(1)}, out; d, in),\ (\mathtt{yes}, out)\} \cup \{(d^{(i+1)}, out; d^{(i)}, in) \mid 1 \leq i \leq m\}\\
&\quad \cup \{(\alpha^{(j+1)}, out; \alpha^{(j)}, in) \mid \alpha \in Val, 1 \leq j \leq m-1\}\\
&\quad \cup \{(\alpha^{(1)}, out; \alpha, in) \mid \alpha \in Val\} \cup \{(a, out; \langle a \rangle, in)\}.
\end{aligned}$$

For an easier understanding, we also present this system in a graphical form, in Figure 3. (We follow here the standard way of representing a P system with

symport/antiport rules, i.e., indicating inside membranes the objects present in the initial configuration of the system, and on the outside side of each membrane the associated rules.)



$$(d^{(i+1)}, out; d^{(i)}, in), 1 \leq i \leq m$$
$$(d^{(1)}, out; d, in)$$
$$(\alpha^{(j+1)}, out; \alpha^{(j)}, in), \alpha \in Val, 1 \leq j \leq m-1$$
$$(\alpha^{(1)}, out; \alpha, in), \alpha \in Val$$
$$(\texttt{yes}, out),\ (a, out; \langle a \rangle, in)$$

Fig. 3. The symport/antiport ωP system from the proof of Theorem 2

The computation of Π starts after introducing the multiset $code(\gamma)$ into the skin membrane, for a given instance γ of the $\texttt{SAT}(n, m)$ problem. The truth-values

which satisfy the first clause bring from the environment the corresponding truth-values without superscripts. Simultaneously, object a is replaced by $\langle a \rangle$, and all other truth-values, corresponding to clauses $C_2, \ldots, C_m$, decrease by one their superscripts. Also the "checker" object d decreases, step by step, its superscript – but starting from $m+1$, hence by one greater than the superscripts of objects associated with C_m. The truth-values without superscripts and object $\langle a \rangle$ "migrate", step by step, towards membrane 1. First, the truth-values which satisfy C_1 (at the same time with $\langle a \rangle$) reach membrane 1, then those which satisfy C_2, and so on.

In membrane 1, objects of the form $\langle aw \rangle$ grow, starting from the "seed" $\langle a \rangle$, with w containing the truth-values which satisfy one by one the clauses. Specifically, if α satisfy clause C_i and it arrives in membrane 1, where we have the object $\langle aw \rangle$ (for $i = 1$ we have $w = \lambda$), with w containing the truth-values of the variables which satisfy all clauses $C_j, 1 \leq j \leq i-1$, then:

(1) if α is in w, then the object $\langle aw \rangle$ is not changed, and α is moved in membrane 0, where it will not survive,

(2) if neither α nor $\neg\alpha$ is in w, then this new truth-value α is added to w, by means of the rule $(\langle aw\alpha \rangle, out; \langle aw \rangle \alpha, in) \in R_0$,

(3) if none of the previous cases holds (i.e., $\neg\alpha$ appears in w), then no reaction takes place – hence both α and $\langle aw \rangle$ will disappear because of the non-permanency condition.

Note the important fact that the threshold assumption is crucial in this operation: each object, whether of the form α or $\langle aw \rangle$, appears in membrane 1 (and in membranes $0, 0'$) in the ω way, sufficient for all rules which can be applied (there is no competition for objects), hence *all* rules are applied simultaneously!

One step after the truth-values corresponding to the last clause, C_m, entered membrane 1, also d moves to membrane 1. It finds here all truth-assignments w which satisfy all clauses. If there is no such truth-assignment, then no reaction takes place in membrane 1 at that time – thus object d disappears and object `yes` is not released from membrane 0. If there is at least one non-empty truth-assignment w, then the rule $(\texttt{yes}, out; \langle aw \rangle d, in) \in R_0$ is used and `yes` is moved out of membrane 0 and from here it starts its way out of the system.

The internal membranes $0, 0'$ have the role of suppliers of objects: because of the non-permanency assumption, only objects which are moved by a rule survive.

If the formula γ is satisfiable, then object `yes` exits the system, otherwise this object remains inside. Let us count now the number of steps necessary to bring out object *yes*. Objects $d^{(i)}$ decrease their superscript from $m+1$ to 1 (hence $m+1$ steps), then $d^{(1)}$ is replaced by d (one more step). In further m steps, d crosses all membranes from region $m+1$ to region 1. Taking `yes` from membrane 0 needs one more step. Crossing all membranes $1, 2, \ldots, m+1$ requires $m+1$ steps. Thus, provided that γ is satisfiable, object `yes` exits the system in $3m+4$ steps.

Because of the rules associated with membrane $0'$, the system never halts, but the answer whether or not the formula γ is satisfiable is obtained in step $3m+4$: γ is satisfiable if and only if in this step we get `yes` out of the system.

Note that we work with precomputed resources: the total alphabet of the system as well as the contents (objects and rules) of membranes $0, 0'$ are exponential (of the order of n^m), the computation of these objects and rules is done in advance, at no cost, but there is no information in the system at the beginning of the computation related to γ different from n and m.

Finally, we notice that the construction is uniform (it starts from the problem itself, $\texttt{SAT}(n, m)$, not from a given instance of the problem), which concludes the proof.

5 Concluding Remarks. Other Cases to Consider

In this paper we continued the study of the effect of transferring to membrane computing the two basic axioms of reaction systems: the *non-permanency assumption* (an object which does not evolve disappears) and the *threshold assumption* (an object either does not appear, or it is present in arbitrarily many copies).

After recalling the results from [12] and [13], we established two new results: in the non-permanency case, SN P systems characterize the semilinear sets of numbers, and symport/antiport systems under the threshold assumption (imposed in only two membranes) can solve SAT in a polynomial time. All these results and the cases which were not yet investigated are displayed in Table 1.

	coop	cat	ncoop	S/A	SN P
Non-permanency	Univ. [13]	?	?	Univ. [13]	$SLIN_1$ Theorem 1
Threshold assumption	Fyper. [12]	?	?	Fyper. Theorem 2	?

Table 1. Cases studied – cases to be studied

Of course, there also are other open problems and research topics. Several of them were also mentioned in the previous sections. Certainly, an interesting question is whether the threshold assumption adds power to SN P systems working under the non-permanency assumption.

A natural research topic is to avoid using precomputed resources in Theorem 2 and instead to construct the exponentially many components of the initial configuration of the symport/antiport P system by using additional features of the system, e.g., using membrane division.

A "dual" research area is a transfer of ideas from membrane computing to reaction systems, but we do not address this issue here – some comments can be found in [13].

Acknowledgements. The work of the first two authors was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200, co-financed by FEDER funds.

References

1. R. Brijder, A. Ehrenfeucht, M. Main, G. Rozenberg: A tour of reaction systems. *International J. Found. Computer Sci.*, 22 (2011), 1499–1518.
2. J. Copeland: Hypercomputation. *Minds and machines*, 12, 461-502 (2002)
3. A. Ehrenfeucht, J. Kleijn, M. Koutny, G. Rozenberg: Qualitative and quantitative aspects of a model for processes inspired by the functioning of the living cell. *Biomolecular Information Processing, From Logic Systems to Smart Sensors and Actuators* (E. Katz, ed.), Wiley-VCH, Weinheim, 2012, 303–322.
4. A. Ehrenfeucht, G. Rozenberg: Reaction systems. *Fundamenta Informaticae*, 75 (2007), 263–280.
5. A. Ehrenfeucht, G. Rozenberg: Events and modules in reaction systems. *Theoretical Computer Sci.*, 376 (2007), 3–16.
6. A. Ehrenfeucht, G. Rozenberg: Introducing time in reaction systems. *Theoretical Computer Sci.*, 410 (2009), 310–322.
7. R. Freund, L. Kari, P. Sosik: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Sci.*, 330 (2005), 251–266.
8. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 279–308 (2006)
9. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–306.
10. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (first circulated as Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
11. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
12. Gh. Păun: Towards fypercomputations (in membrane computing). *Languages Alive. Essays Dedicated to Jurgen Dassow on the Occasion of His 65 Birthday* (H. Bordihn, M. Kutrib, B. Truthe, etcs.), LNCS 7300, Springer, Berlin, 2012, 207–221.
13. Gh. Păun, M.J. Pérez-Jiménez: Towards bridging two cell-inspired models: P systems and R systems. *Theoretical Computer Sci.*, 429 (2012), 258–264.
14. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.
15. M.J. Pérez-Jiménez: Computational complexity theory in membrane computing (Invited talk). *New Frontiers in Informatics. 24th Annual Conference of Academia Europaea*, Bergen, Norway, September 11, 2012.
16. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.
17. The P Systems Website: `http://ppage.psystems.eu`.