Towards Bridging Two Cell-Inspired Models: P Systems and R Systems

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

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Summary. We examine, from the point of view of membrane computing, the two basic assumptions of reaction systems, the "threshold" and "no permanence" ones. In certain circumstances (e.g., defining the successful computations by local halting), the second assumption can be incorporated in a transition P system or in a symport/antiport P system without losing the universality. The case of the first postulate remains open: the reaction systems deal, deterministically, with finite sets of symbols, which is not of much interest for computing; three ways to introduce nondeterminism are suggested and left as research topics.

1 Introduction

The aim of this note is to bridge two branches of natural computing inspired from the biochemistry of a living cell, *membrane computing* (see, e.g., [11], [12], [13], and the domain website from [15]) and the recently introduced *reaction systems* area – see [2], [3], [4], [5], [6].

Both areas deals with populations of reactants (molecules) which evolve by means of reactions, with several basic differences. Most of these differences are not mentioned here (e.g., the compartmental structure of models – P systems – in membrane computing versus the missing of membranes in reaction systems – we also call them *R systems* –, the focus on evolution, not on computation, in reaction systems, the unique form of rules in reaction systems and so on), and we recall the two basic ones in the formulation from [2]:

The way that we define the result of a set of reactions on a set of elements formalizes the following two assumptions that we made about the chemistry of a cell:

(i) We assume that we have the "threshold" supply of elements (molecules) – either an element is present and then we have "enough" of it, or an element is

not present. Therefore we deal with a qualitative rather than quantitative (e.g., multisets) calculus.

(ii) We do not have the "permanence" feature in our model: if nothing happens to an element, then it remains/survives (status quo approach). On the contrary, in our model, an element remains/survives only if there is a reaction sustaining it.

Passing from multisets, which are basic in P systems, to sets (actually, to multisets with an infinite multiplicity of their elements) is a fundamental assumption, which changes completely the approach; for instance, we can no longer define computations with the result expressed in terms of counting molecules: the total set of molecules is finite, any molecule is either absent or present in infinitely many copies. Moreover, the behavior of a reaction system is deterministic, from a set of symbols we precisely pass to a unique set of symbols (hence the behavior of a reaction system can be described by a graph of outdegree one, having the nodes marked with subsets of the total set of molecules). How to bridge at this level the two research areas (defining computations in reaction systems or working with multisets with infinite multiplicity of each element in P systems) remains as a research topic. Here we only propose three ways to introduce nondeterminism in reaction systems, so that more interesting computation (evolution) graphs can be obtained: providing tables of rules, considering also molecules with a finite multiplicity, and considering a threshold on the number of rules which can use simultaneously molecules of a given type.

P systems with sets were also considered in [10], mainly from the semantics (via Petri nets) point of view.

The second assumption of the reaction systems theory is much easier to handle in terms of membrane computing. The immediate idea is to simply remove any element which does not evolve by means of a reaction; somewhat equivalently, if we want to preserve an object a which is not evolving, we may provide a dummy rule for it, of the type $a \to a$, changing nothing.

Still, many technical problems appear in this framework. The presence of such dummy rules makes the computation endless, while halting is the "standard" way to define successful computations in membrane computing. Moreover, the rules are nondeterministically chosen, hence the dummy rules can interfere with the "computing rules".

While the second difficulty is a purely technical one, the first one can be overpassed by considering other ways of defining the result of a computation in a P system, and there are many suggestions in the literature. We consider here three possibilities: (i) the *local halting* of [8] (the computation stops when at least one membrane in the system cannot use any rule), (ii) *signal-objects* (the result consists of the number of objects in a specified membrane at the moment when a distinguished object appears in the system), (iii) *signal-events* (the result consists of the number of objects in a specified membrane at the moment when a distinguished rule is used in the system). Such signals were considered in various papers; we refer here only to [9].

All these possibilities are checked both for transition and for symport/antiport P systems – with some cases still remaining open (the most important one is that of catalytic P systems).

2 Basic Definitions

For the sake of completeness, we recall here a few elementary notions about reaction systems and P systems.

The language theory notations are standard. An alphabet is a finite and nonempty set. For an alphabet V, by V^* we denote the set of all strings over V, including the empty string, denoted by λ. The set of nonempty strings over V is denoted by V^+. The length of a string $x \in V^*$ is denoted by $|x|$. The multisets over a finite set S are represented by strings in S^*; a string and all its permutations represent the same multiset. (The Parikh mapping of a string representing a multiset indicates the multiplicity of each object in the multiset.)

In the proofs from Section 4 we will use the characterization of recursively enumerable sets of numbers (sets of numbers computable by Turing machines; their family is denoted by NRE, reminding the fact that these sets ate length sets of recursively enumerable languages) by means of *register machines*; such a device is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- l_i : (`ADD`$(r), l_j, l_k$) (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- l_i : (`SUB`$(r), l_j, l_k$) (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- l_h : `HALT` (the halt instruction).

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by labels (and made possible by the content of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, i.e., they characterize the family NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its content.

2.1 Reaction Systems

We recall here some elementary notions and notation about reaction systems, as available in the few papers already published in this area – see again the titles mentioned at the beginning of the Introduction.

Let S be an alphabet (its elements are called *molecules* or, simply, symbols). A *reaction* (in S) is a triple $a = (R, I, P)$, where R, I, P are nonempty subsets of S such that $R \cap I = \emptyset$. R is the *reactant set of* a, I is the *inhibitor set of* a, and P is the *product set of* a. R, I, P are also denoted R_a, I_a, P_a. We denote by $rac(S)$ the set of all reactions in S.

If $T \subseteq S$ and $a \in rac(S)$, then a *is enabled by* T if $R_a \subseteq T$ and $I_a \cap T = \emptyset$, and then the *result of* a *on* T, denoted by $res_a(T)$, is defined by $res_a(T) = P_a$. If a is not enabled by T, then $res_a(T) = \emptyset$.

If A is a finite set of reactions, then *the result of* A *on* T is defined by $res_A(T) = \bigcup_{a \in A} res_a(T)$.

Then, a *reaction system* (we also call it an R *system*) is an ordered pair $\sigma = (S, A)$, where S is an alphabet and $A \subseteq rac(S)$.

Note in the definition of the result of a set A of reactions on a set T of molecules the occurrence of the two assumptions mentioned in the Introduction: a molecule can evolve by means of several reactions (or can inhibit several reactions if it appears in inhibitor sets), hence the multiplicity of each molecule is unbounded, while all molecules present at a given time "disappears", after the reactions we continue with the set of molecules produced by the reactions.

2.2 P Systems

We introduce first the class of transition P systems, closer in their definition to reaction systems. Some familiarity of the reader with the elementary notions of membrane computing is assumed, e.g., from [12], [13].

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, a *region* is associated, the space delimited by it and the inner membranes, if any. A membrane structure can be represented by a rooted tree or by an expression of labeled parentheses (with a unique external parenthesis, associated with the skin).

Given an alphabet O of *objects*, a multiset-rewriting rule (over O; we also say *evolution rule*) is a pair (u, v), written in the form $u \to v$, where u and v are multisets over O (given as strings in O^*). The rules are classified according to the complexity (of their left hand side). A rule with at least two objects in its left hand side is said to be *cooperative*; a particular case is that of *catalytic* rules, of the form $ca \to cv$, where c is a catalyst which assists the object a (which is not a catalyst) to evolve into the multiset v (where no catalyst appears); rules of the form $a \to v$, where a is an object, are called *non-cooperative*.

The rules can also have associated promoters or inhibitors, objects whose presence make possible the use of a rule (but are not modified by the rule application), respectively, can forbid the application of the rule. Also, a *priority* relation can be considered, in the form of a partial order relation among the set of rules in a membrane; a rule can be used only if no rule of a higher priority can be used. Finally, we mention the *dissolution* operation: a rule can be of the form $u \to v\delta$ and, when used, the membrane in which it is applied is "dissolved", its objects become elements of the immediately higher membrane (and its rules disappear, as being associated with the "reactor" defined by the membrane). We do not enter here into details – in general, such additional controls on using the rules are rather useful (and powerful) in "programming" the work of a P system.

Now, a *transition P system* (of degree m) is a construct

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

where O is the alphabet of objects, μ is the membrane structure (with m membranes), given as an expression of labeled parentheses, $w_1, \ldots, w_m$ are (strings over O representing) multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of evolution rules associated with the regions of μ, and i_{in}, i_{out} are the labels of input and output membranes, respectively. If the system is used in the generative mode, then i_{in} is omitted, and if the system is used in the accepting mode, then i_{out} is omitted. If the system is a catalytic one, then a subset C of O is specified, containing the catalysts. The number m of membranes in μ is called the *degree* of Π.

The rules in sets R_i are of the form $u \to v$, as specified above, with $u \in O^+$, but with the objects in v also having associated *target indications*, i.e., $v \in (O \times \{here, out, in\})^*$. After using a rule $u \to v$, the objects in u are consumed, and those in v are produced; if $(a, here)$ appears in v, then a remains in the same compartment of the system where the rule was used, if (u, out) is in v, then the object a is moved immediately in the region surrounding the compartment where the rule was used (this is the environment if the rules is used in the skin region), and if (a, in) is in v, then a is sent to one of the inner membranes, nondeterministically chosen (if there is no membrane inside the membrane where the rule is meant to be applied, then the use of the rule is forbidden). The indication *here* is omitted, we write a instead of $(a, here)$.

The rules are used in the nondeterministic maximally parallel manner: in each membrane, a multiset of rules is applied such that there is no larger multiset of rules which is applicable in that membrane.

In the generative mode, the result of a computation consists of the number of objects in membrane i_{out} in the moment when the computation halts, i.e., no rule can be applied in any membrane of the system. In the accepting mode, a number is introduced in the membrane i_{in}, in the form of the multiplicity of a given object, and, if the computation halts, then this number is accepted. A P system can also be used in the computing mode, with a number introduced in membrane i_{in} and the result obtained in membrane i_{out}, in the moment when the computation halts.

In what follows, we only deal with generating P systems. One knows that catalytic P systems are Turing equivalent, they compute all recursively enumerable sets of natural numbers (i.e., they characterize NRE), but non-cooperative P systems compute only semilinear sets of numbers. Details can be found in the references given at the beginning of the Introduction.

Another much investigated class of P systems is that of *symport/antiport P systems*. These systems are not based on reaction rules, but on biological operations of passing coupled molecules across membranes.

We can formalize these operations by considering *symport* rules of the form (x, in) and (x, out), and *antiport* rules of the form $(z, out; w, in)$, where x, z, and w are multisets of objects.

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 all components $O, \mu, w_1, \ldots, w_m, i_{in}, i_{out}$ are as in a P system with multiset rewriting rules, $E \subseteq O$, and $R_1, \ldots, R_m$ are finite sets of symport/antiport rules associated with the m membranes of μ. The objects of E are supposed to be present in the environment of the system with an arbitrary multiplicity. (Note that the symport/antiport rules do not change the number of objects, but only their place, that is why we need a supply of objects in the environment; this supply is inexhaustible, i.e., does not matter how many objects are introduced in the system, arbitrarily many still remain in the environment.)

As above, the rules are used in the nondeterministic maximally parallel manner: we choose nondeterministically multisets of rules associated with each membrane and such an m-tuple of multisets is applied if for no membrane a rule can be added to the associated multiset still having the enlarged m-tuple of multisets applicable. We define transitions, computations, and halting computations in the usual way. The number of objects present in region i_{out} in the halting configuration is said to be computed by the system by means of that computation; the set of all numbers computed in this way by Π is denoted by $N(\Pi)$. Accepting and computing symport/antiport P systems are defined in the natural manner.

It is known that symport/antiport P systems (with a small number of membranes and with rules of a low complexity) characterize NRE.

Note that in the previous definitions multisets play a crucial role, objects not evolving by a rule remain unchanged, and that always successful computations are defined by halting.

3 Computing with Reaction Systems

Starting from a reaction system $\sigma = (S, A)$, we can consider a "generative device" $\gamma = (S, A, w_0)$, where w_0 is a subset of S, an "axiom set". (We denoted the starting set by a small letter, like a string, in the multiset sense, because we will need such

an approach below, e.g., when part of molecules will be considered in the multiset sense.) Then, we can obtain a sequence $w_0 \Longrightarrow_A w_1 \Longrightarrow_A w_1 \Longrightarrow_A \ldots$, where $w_{i+1} = res_A(w_i), i \geq 0$.

Two basic observations: (i) this sequence is unique, because the passage from a set of molecules to the next one is deterministic, and (ii) for all $i \geq 0$ we have $w_i \subseteq S$. Therefore, if we associate a label to each subset of S, then a sequences as above is either finite (at some moment, no rule can be applied, all elements vanishes, hence we end with the label of the empty set), or the sequence is infinite and then it can be described by a string of the form uv^ω: after a finite path among subsets of S, we enter a cycle which goes forever.

In terms of graphs, the relation $\Longrightarrow_A$ defines a graph $G_S(A) = (2^S, \Longrightarrow_A)$ of outdegree (at most) one (the outdegree can be zero, but this is a trivial case). Computations in $\gamma = (S, A, w_0)$ can then be followed along the paths in $G_S(A)$ starting in the node w_0.

We do not have here too much from a computability point of view, even if we consider the graph itself as the result of the computation (the number of graphs $G_S(A)$ is bounded, because of the finiteness of S). The dramatic restriction here is the deterministic behavior of a reaction system, that is why we propose here three possibilities to get a nondeterministic device.

The first natural idea is to consider a *tabled* reaction system, in the form $\gamma = (S, A_1, A_2, \ldots, A_n, w_0)$ where $A_i, 1 \leq i \leq n$, are sets of reactions over S (called tables). Like in an E0L system (see, e.g., [14]), in a step of a computation we can nondeterministically choose the table to use, hence branching is possible. In this case, we can also introduce halting as a criterion for defining successful computations: a halt table can be considered, for instance, with rules of the form $a \to a'$ for all $a \in S$, such that no rule exists for a', hence in the next step all (primed) molecules disappear.

Another idea, at the bridge of membrane computing and reaction systems, is to consider a subset $C \subseteq S$ of molecules for which the multiplicity matters, and having finite multiplicities. Then we move towards usual P systems (cooperative, with inhibitors, hence rather powerful). The elements of C are counted when applying the rules, those in $S - C$ not. The nondeterminism appears now when using copies of elements in C, if more rules than such objects can be applied.

Finally, without modifying the components of a computing reaction system $\gamma = (S, A, w_0)$, we can provide the nondeterminism by introducing a general threshold on the number of rules which can use the same molecule, hence having a system of the form $\gamma = (S, A, w_0, k)$, where k is the threshold. This is similar to the previous case, taking $C = S$, which is like working with multisets, but with the same multiplicity for all objects (only at most k copies of each object can evolve, the others are removed, hence we can assume that the multiplicity is exactly k for each object). The nondeterminism appears again when choosing the rules which compete for the same objects. We have a usual P system, but dealing with finite populations of objects: if only k rules are used for each molecule, only finitely many

rules are used, all existing objects are consumed or they vanishes and a bounded number of objects are produced.

In the second case, the multiplicity of objects in C can increase arbitrarily, but in the other two cases we again deal with a finite computation graph (but not of an outdegree bounded in advance).

All these three possibilities remain to be investigated: properties of the obtained graphs, possible links with computing devices from formal language and automata theory, influence of the introduced parameters (number of tables, cardinality of C, threshold k), possible hierarchies.

Of course, another research topic is to find other ways of building a (string or graph) computing device in terms of reaction systems.

4 P Systems without the "Permanence" of Objects

Let us now move to membrane computing, and borrow from reaction systems area the assumption that an object which is not involved in a rule does not pass to the next configuration. Then, we cannot define the result of a computation by halting, because in a halting step all objects vanish. Similarly, it is not enough to add dummy rules of the form $a \to a$ (in transition systems), because this time the computation never halts. Thus, we have to define successful computations by other conditions – and we consider here the three possibilities recalled in the Introduction: local halting, signal-objects, signal-events. The definitions are straightforward, we pass directly to examine the power of P systems endowed with such conditions.

4.1 The Case of Transition P Systems

Let us consider a register machine $M = (m, H, l_0, l_h, I)$, as introduced at the beginning of Section 2. We first construct a transition P system $\Pi = (O, \mu, w_1, w_2, R_1, R_2, 1)$, aiming to simulate the machine M, and then we discuss modes of defining the result of a computation in Π. We take:

$$
\begin{aligned}
O &= \{a_i, a'_i \mid 1 \leq i \leq m\} \cup \{l, l', l'', l''', l^{iv} \mid l \in H\} \cup \{b, c, \#\},\\
\mu &= [\,[\quad]_2\,]_1,\\
w_1 &= l_0,\ w_2 = b,\\
R_1 &= \{l_i \to l_j a_r,\\
&\qquad l_i \to l_k a_r \mid l_i : (\texttt{ADD}(r), l_j, l_k) \in I\}\\
&\quad \cup \{a_1 \to a_1\} \cup \{a_s \to a_s a'_s \mid 2 \leq s \leq m\}\\
&\quad \cup \{l_i \to l'_i l''_i,\\
&\qquad l'_i a_r \to l'''_i,\\
&\qquad l'_i a'_r \to (\#, in),
\end{aligned}
$$

$$
\begin{aligned}
& l_i'' \to l_i^{iv}, \\
& l_i^{iv} \to l_k, \\
& l_i''' \to (\#, in), \\
& l_i^{iv} l_i''' \to l_j \mid l_i : (\mathtt{SUB}(r), l_j, l_k) \in I\} \\
\cup\ & \{l_h \to (l_h, in)\}, \\
R_2 = \{& b \to b,\ \# \to \#,\ l_h b \to c\}.
\end{aligned}
$$

This system works as follows. The contents of each register r is represented by the number of occurrences of objects a_r in the skin region of Π. In each step, each of these objects is reproduced, hence their number is never decreased; moreover, objects $a_r, r \neq 1$, also produce "twin objects" a_r', which disappear in the next step (one copy is used in simulating SUB instructions, as we will see below). Object b evolves forever in membrane 2. One of our goals is to define the end of a computation in Π by local halting, namely, by halting the evolution of membrane 2. This can happen only in the presence of the halt label of M, and without introducing the trap-object #.

We start with label l_0 in membrane 1. In general, when a label l_i is present in membrane 1, the respective instruction of M is simulated.

The simulation of an ADD instruction is obvious. Assume that l_i is the label of a SUB instruction, $l_i : (\mathtt{SUB}(r), l_j, l_k)$. We use the rule $l_i \to l_i' l_i''$. At the same time, all objects a_s' disappear and all objects a_s are replaced by $a_s a_s'$, $2 \leq s \leq m$; objects a_1 remains always unchanged during simulating a SUB instruction (remember that M never decreases register 1). In the next step, l_i'' is replaced by l_i^{iv}, while l_i' has two possibilities. If a copy of a_r is present (hence register r is not empty), then also a_r' is present. If the rule $l_i' a_r \to l_i'''$ is used, then a_r' disappear, and this is the correct continuation – in the next step, the rule $l_i^{iv} l_i''' \to l_j$ is used, introducing the label of the next instruction to simulate. If, instead of $l_i' a_r \to l_i'''$, the rule $l_i' a_r' \to (\#, in)$ is used, then the trap-object # is introduced in membrane 2, and it will evolve here forever. If the register r is empty, hence no object a_r and a_r' is present, then l_i''' is not introduced, l_i' disappears. In the next step l_i^{iv} has to evolve by means of the rule $l_i^{iv} \to l_k$, the correct continuation in the register machine. If this rule is used also in the presence of l_3''' (hence in case the register r was nonempty), then we have to use the rule $l_i''' \to (\#, in)$.

In this way, the instructions of M are correctly simulated. When the halt label l_h is introduced in M, this object is moved to membrane 2. If the only object present here is b, then the computation in membrane 2 can halt by means of $l_h b \to c$. If also # is present, then the computation in membrane 2 continues forever. The number of objects a_1 in membrane 1 at the moment of halting membrane 2 gives the result of the computation. (Remember that all registers of M except the first one are empty in the end of computations in M.)

The previous construction can be slightly modified in order to mark the end of the computation by means of signal objects or events instead of local halting. For instance, if we replace the rule $\# \to \#$ of R_2 with $\# \to \delta$, then membrane

2 is dissolved, the rule $l_h b \to c$ cannot be used. Thus, the signal can be either the object c or the use of the rule $l_h b \to c$. When one of these signals appears in membrane 2, the number of copies of a_1 in membrane 1 is the result of the computation. If # was introduced, then these signals never appear.

We conclude with the assertion-theorem that *transition P systems of degree 2, using cooperative rules, without the "permanence" of objects, are computationally complete.*

An interesting *open problem* in this framework is the case of catalytic P systems, known to be universal in the "permanence" assumption (see, e.g., [7]).

4.2 The Case of Symport/Antiport P Systems

The case of symport/antiport systems just "recodes" the previous construction, but, because for these systems we do not have the dissolution operation (it can be introduced, in a natural way, but this was not done up to now, hence we do not consider it here), only the case of local halting is considered.

Take again a register machine $M = (m, H, l_0, l_h, I)$. We construct the symport/antiport P system $\Pi = (O, \mu, w_1, w_2, E, R_1, R_2, 1)$ with the same alphabet of objects and membrane structure as in the previous subsection, but with $w_1 = bl_0,\ w_2 = b,\ E = O$, and with the rules as specified in Figure 1 – instead of a formal definition, we give now the graphical representation of the system.

The functioning of this system is very much similar to the functioning of the system in the previous subsection, hence we do not describe it in details (membrane 2 halts only when # is not present and l_h moves outside the system the object b from the skin region).

The case of defining the result of a computation by means of signals – objects or events (using a specified rule) – remains as an *open problem.* (Considering a priority relation on each set of rules can easily solve this problem.) The previous symport/antiport P system contains antiport rules of sizes (2, 1) and (1, 2), which is "large" for universality results in the case when objects are persistent (see, e.g., [1]). Can the size of rules be decreased also in the case discussed here?

5 Final Remarks

Although there are so many similarities and differences between membrane computing (P systems) and reaction systems (R systems), up to our knowledge, so far there is no bridging investigation, in spite of the fact that this research topic was formulated several times in the membrane computing community (e.g., during the yearly Brainstorming Weeks on Membrane Computing). This is a natural and surely fruitful area to explore, especially in checking the influence of basic postulates of one domain in another one and in borrowing notions and research issues from a domain to another one. The present paper is only a first step in this direction, examining the two basic postulates of reaction systems: working with

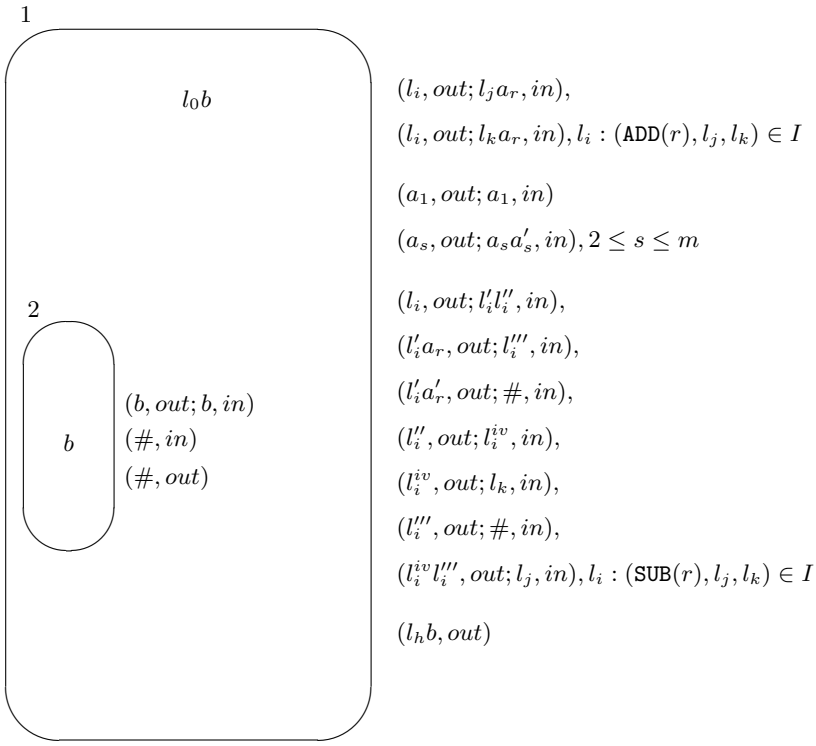

Fig. 1.

molecules whose multiplicity is not counted (it is considered infinite) and removing from the system molecules which do not evolve by reactions. Many open problems and research topics are formulated.

Acknowledgements. Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Some optimal results on symport/ antiport P systems with minimal cooperation. In *Cellular Computing (Complexity Aspects)* (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, eds.), ESF PESC Exploratory Workshop, Fénix Editorial, Sevilla, 2005, 23–36.
2. A. Ehrenfeucht, G. Rozenberg: Basic notions of reaction systems, *Proc. DLT 2004* (C.S. Calude, E. Calude, M.J. Dinneen, eds.), LNCS 3340, Springer, 2004, 27–29.
3. A. Ehrenfeucht, G. Rozenberg: Reaction systems. *Fundamenta Informaticae*, 75 (2007), 263–280.

4. A. Ehrenfeucht, G. Rozenberg: Events and modules in reaction systems. *Theoretical Computer Sci.*, 376 (2007), 3–16.
5. A. Ehrenfeucht, G. Rozenberg: Introducing time in reaction systems. *Theoretical Computer Sci.*, 410 (2009), 310–322.
6. A. Ehrenfeucht, G. Rozenberg: Reaction systems. A model of computation inspired by biochemistry. *Proc. DLT 2010* (Y. Gao et al., eds.), LNCS 6224, Springer, 2010, 1–3.
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. R. Freund, M. Oswald: Partial halting in P systems. *Intern. J. Foundations of Computer Sci.*, 18 (2007), 1215–1225.
9. P. Frisco: *Computing with Cells. Advances in Membrane Computing*. Oxford University Press, 2008.
10. J. Kleijn, M. Koutny: Membrane systems with qualitative evolution rules, *Fundaenta Informaticae*, to appear.
11. 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`).
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
13. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
14. G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
15. The P Systems Website: `http://ppage.psystems.eu`.