A Guide to Membrane Computing

Gheorghe PĂUN[1]
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Grzegorz ROZENBERG
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
and
Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309, U.S.A.
E-mail: rozenber@liacs.nl

Abstract. Membrane systems are models of computation which are inspired by some basic features of biological membranes. In a membrane system multisets of objects are placed in the compartments defined by the membrane structure, and the objects evolve by means of "reaction rules" also associated with the compartments, and applied in a maximally parallel, nondeterministic manner. The objects can pass through membranes, the membranes can change their permeability, they can dissolve, and they can divide. These features are used in defining transitions between configurations of the system, and sequences of transitions are used to define computations. In the case of symbol-objects, we compute a set of numbers, and in the case of string-objects we compute a set of strings, hence a language. Many variants of such computing devices (now called P systems) have already been investigated. Most of them are computationally universal, i.e., equal in power to Turing machines. Systems with an enhanced parallelism are able to trade space for time and solve in this way (at least in principle), by making use of an exponential space, intractable problems in a feasible time.

The present paper presents the basic ideas of computing with membranes and some fundamental properties (mostly concerning the computational power and efficiency) of P systems of various types[2].

1 Introduction

The basic function of biological membranes is to *define compartments* and to *relate compartments to their environment*, including neighboring compartments. For instance, the plasma membrane (see, e.g., [2]) ensures that certain substances (molecules) stay within

[1]Work supported by a grant of NATO Science Committee, Spain, 2000–2001.

[2]The current bibliography of membrane computing can be found at the web address
http://bioinformatics.bio.disco.unimib.it/psystems

(do not escape from) the cell, while other substances, e.g., toxic molecules, stay out of the cell. Moreover, membranes allow certain molecules to pass through: e.g., waste products to leave, and certain nutrients to enter. Also, membranes form a communication structure, allowing messages (signals) to be received or to be transmitted by the enclosed space. This communication is crucial for establishing multicellular communication and hence for establishing multicellular organization (see, e.g., [26]). This compartmentalization by membranes, with each enclosed area having its own set of molecules and (enzymes enhancing) reactions, with the transport of molecules and (hence) the communication through membranes, is the paradigm underlying *membrane systems* (see, e.g., [40] and Chapter 3 of [10]).

It must be stressed that membrane systems (also called *P systems*) are *not* intended to model the functioning of biological membranes. Rather, we explore the computational nature of various features of membranes, i.e., we investigate how such features can be used in a model of computation. To this aim, we abstract from a number of principles underlying the functioning of biological membranes, and use this abstraction to construct a novel model of computing. Such an approach is typical for the area of *Natural Computing*, where one studies all kinds of computing inspired by (or gleaned from) nature.

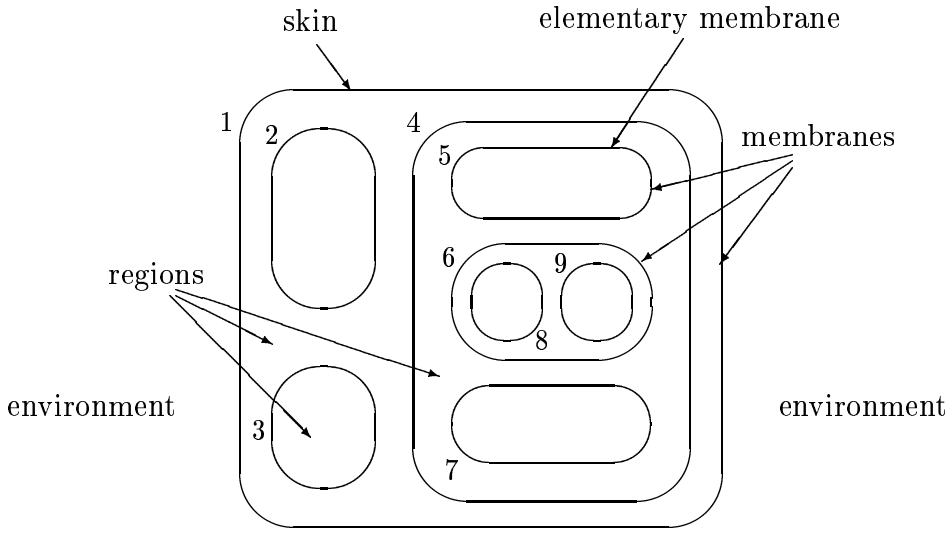

Figure 1: A membrane structure

The *membrane structure* of a P system is a hierarchical arrangement of membranes (understood as three dimensional vesicles), embedded in a *skin* membrane, the one which separates the system from its *environment*. A membrane without any membrane inside is called *elementary*. Each membrane defines a *region*. For an elementary membrane this is the space enclosed by it, while the region of a non-elementary membrane is the space in-between the membrane and the membranes directly included in it. Figure 1 illustrates these notions. We label membranes (by positive integers in Figure 1) in order to be able to address them in programming computations by membrane systems. Since each region is delimited ("from the outside") by a unique membrane, we will use the labels of membranes to also identify (label) the regions they delimit. We will use the obvious

terminology here – thus, for example, we say that membrane 8 is directly contained in membrane 6 (or that membrane 6 directly contains membrane 8).

Each region contains a multiset of *objects*, and a set of (*evolution*) *rules*. The objects are represented by symbols from a given alphabet. Typically, an evolution rule from region r is of the form $ca \to cb_{in_j} d_{out} d_{here}$, and it "says" that a copy of the object a, in the presence of a copy of the *catalyst* c (this is an object which is never modified, it only assists the evolution of other objects), is replaced by a copy of the object b and two copies of the object d. Moreover, the copy of b has to "immediately" enter the inner membrane of region r labeled by j (hence to enter region j), a copy of object d is sent out through the membrane of region r, and a copy of d remains in region r. Note that the considered evolution rule can be applied in the region r only if this region includes the membrane j.

Note that (syntactic) evolution rules are stated in terms of objects while their implemenation (execution) is done using *copies of objects*: at a given moment the given region may contain many copies (a multiset) of a given object. In order to simplify descriptions of computations in membrane systems, we will often use the term "object" rather than "a copy of an object", but, bearing in mind the above principle, the real meaning will always be clear from the context of considerations.

Membrane systems are synchronous, in the sense that a global clock is assumed, i.e., the same clock holds for all regions of the system. In each time unit a transformation of a *configuration* of the system takes place by applying the rules in each region, in a *nondeterministic* and *maximally parallel manner*. This means that the objects to evolve and the rules governing this evolution are chosen in a nondeterministic way; this choice is "exhaustive" in the sense that, after the choice was made, no rule can be applied anymore in the same evolution step (there are not enough objects available anymore for any rule to be applied now – this is the maximality of application).

It is instructive to see a single step transforming a configuration of the system as a "macro-step" consisting of several "micro-steps" performed after each other. Consider a region r of the system. First, we assign (occurrences of) objects from r to rules from r, nondeterministically choosing rules and objects until no further assignment is possible (note that the multiplicity of objects present in r is crucial in this micro-step). Then, all these "assigned" objects are removed from the current multiset of objects in r, and (occurrences of) all objects specified by the right hand sides of the chosen rules are added to this multiset, together with their "transfer commands": in_j, *out, here*. Now, all transfers indicated by commands in_j and *out* are executed (if a copy of an object is introduced in the skin region, i.e., the region delimited by the skin membrane, and its transfer command is *out*, then it will be sent out of the system, to the environment, and it never "comes back"), and copies of objects with the transfer command *here* remain in region r. Finally, the transfer commands (subscripts) are removed, and a "macro-step" is completed for r. Since all regions are processed "simultaneously" (with all micro-steps performed synchronously), this completes the global macro-step.

In this way, one gets *transitions* between the configurations of the system. A sequence of transitions is called a *computation*. A configuration is *halting*, if no rule is applicable in any region (nothing can happen anymore). A computation is *halting* if it reaches a halting configuration. We consider only halting computations. The *result* of a (halting) computation is the *number* of objects sent (through the skin membrane) to the environment

during the computation.

Many modifications/extensions of this very basic model described above are discussed in the literature. We will now briefly discuss two additional features that will be used in the basic model of membrane systems considered in this paper.

The first one is a priority relation among rules. This means that in each region a partial order relation on the set of rules in this region is given – then, a rule can be chosen (to process a multiset of objects) in a given step only if no rule of a higher priority is applicable.

Another "control device" for P systems considered in the literature is a modification of membrane permeability. Thus, the membranes can be *dissolved* (the objects of a dissolved membrane remain in the region surrounding it, while the rules are removed; the skin membrane cannot be dissolved), or made *impermeable* (no object can pass through such a membrane).

There are two standard ways of investigating the influence of various features of P systems: (1) to consider their *computational power/competence* – e.g., are P systems using given features computationally universal (hence equivalent to Turing machines)?, and (2) to consider their *computational complexity* (are P systems able to make use of their intrinsic parallelism and solve hard problems, e.g., NP-complete problems, in a feasible time?). These two topics will be considered in detail in this paper.

As the reader can realize from the previous discussion, and as it will be obvious also below, membrane computing is related to various areas, such as formal language theory, Lindenmayer systems, vector addition systems, multiset processing (e.g., in the sense of the Gamma language, [4]), the Chemical Abstract Machine of [6], but it is also significantly different from all these. In particular, we stress the similarity with and, mainly, the differences from the Chemical Abstract Machine, which also uses a membrane structure and deals with multisets of objects, but involves quite different operations, both when acting on objects and on membranes, and has a quite different goal (to simulate distributed processes, while here we deal with computing in the Turing sense, of computing functions). It is also worth noting that we look here for "minimalistic" models, using a minimal number of simple features, which is not the case in either [4] or [6]. For example, the standard rule in the Gamma language is of a maximal generality, it has the form $(u \rightarrow v; \pi)$, where u and v are multisets, and π is a predicate on the family of multisets; the rule $u \rightarrow v$ is applied for "rewriting" a multiset w only if $P(w) = true$.

2 Bio-membranes; Structure and Functions

In this section we describe in more detail some of the basic features of the plasma membranes. We have chosen those features that are of interest from a computational point of view and from which we will abstract the (mathematical) features of our computing model.

A cell has a complex structure, with several compartments delimited inside the main membrane by several inner membranes: the nucleus, the Golgi aparatus, various vesicles, etc. In principle, all these membranes fulfill the same main roles: they are *separators* and *filters*. We now recall some facts concerning the structure and the functioning of the

plasma membrane (see, e.g., [2] and [26]) which will be relevant in the sequel of this paper.

The currently accepted model of the membrane structure is the so-called *fluid-mosaic model*, proposed in 1972 by S. Singer and G. Nicolson. According to this model, a membrane is a phospholipid bilayer in which protein molecules (as well as other molecules, such as cholesterol, steroids and others) are totally or partially embedded – this is illustrated in Figure 2.

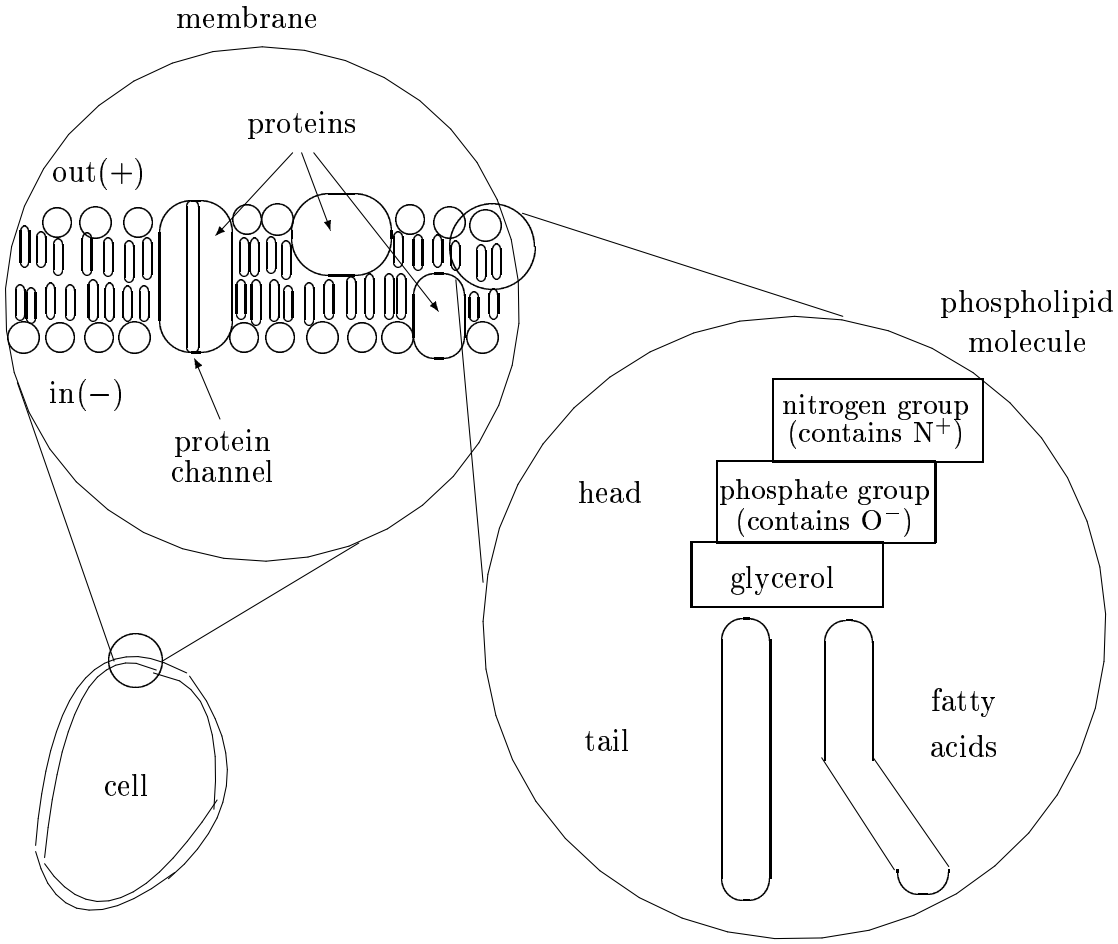

Figure 2. The structure of the plasma membrane

The phospholipid molecules are composed of two main parts: a polar *head* and a non-polar *tail*. The head is composed of a phosphate group and a nitrogen group, the tail consists of two fatty acid chains; the head is bonded to the tail by a glycerol. Consequently, the heads of the molecules in the two layers are hydrophylic, while the tails are hydrophobic. This explains the arrangement of heads against the aqueous solutions from the inner region (plasma) and from outside the cell, as well as the difficulty of passing water through a membrane. Moreover, the polar heads lead to polarizations of the two sides of the membrane: positive charge in the outside layer and negative in the inner layer of molecules. This facilitates the exit of negative ions and the entrance of positive ions.

The (plasma) membrane is only partially permeable. For instance, small noncharged molecules, particularly if they are lipid soluble, cross the membrane almost freely. Larger molecules can only cross a membrane if they are assisted, while charged ions pass selectively from a region to another one.

The transmembrane transfer of molecules can take place in a *passive* manner, e.g., by diffusion towards the region of lower concentration, and in an *active* (mediated) manner. The most important active membrane transfer is done by *protein channels* present in various numbers in membranes. For instance, water (which otherwise cannot pass through the hydrophobic barrier of the tails of the phospholipidic molecules) can pass through such channels.

Actually, there are two main types of protein channels, some which just select the moving objects by their size, and others, the so-called *carrier proteins*, which interact with specific molecules (perhaps also modifying them) when helping them to cross the membrane.

Other important functions of membrane proteins are the *catalytic* activity (certain reactions can take place only in the presence of certain enzymatic proteins), *recognition* and *binding* activities (certain proteins recognize certain molecules or even catch them and keep them bound to the membrane).

Another important aspect is the way the neighboring cells establish protein channels for inter-cellular communication: due to the fact that the phospholipid molecules can move on the membrane surface (that is why the model is called the "fluid-mosaic" one), when two membranes touch each other, their proteins can "look for each other"; when two proteins come close enough, they bind to each other and establish a unique channel through the two membranes. In this way, a complex communication network can be established among cells. If one of the cells is invaded by "undesired" molecules, then the cell isolates itself from the neighboring cells by closing the passage channels – they may be re-opened again, once the emergency situation has been resolved.

We conclude this section by stressing once again that biological membranes provide "a protected and well equipped" space (a kind of a natural reaction tube) within which chemical reactions can take place. Correspondingly, membranes in a P system provide space (and objects) for computations.

3 The Basic Model

We move now to a more formal presentation of the membrane-based computing paradigm, by introducing one of the basic variants of P systems, followed by an example.

A membrane structure is pictorially represented by an Euler-Venn diagram (like the one in Figure 1); it can be mathematically represented by a tree, or by a corresponding string of matching parentheses. For instance, the membrane structure from Figure 1 is represented by the following parentheses expression:

$$[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ [_8\ ]_8\ [_9\ ]_9\ ]_6\ [_7\ ]_7\ ]_4\ ]_1.$$

Since the membranes have labels, the pairs of corresponding parentheses also have labels. It should be noted that the same membrane structure may be represented by different

parenthetic expressions (the order of neighboring membranes placed in the same upper membrane does not matter).

For the basics of formal language theory we refer the reader to, e.g., [53] (as a matter of fact, our use of formal language theory in this paper is quite limited). We use V^* to denote the set of all strings over the alphabet V (we consider only finite alphabets). For $a \in V$ and $x \in V^*$ we denote by $|x|_a$ the number of occurrences of a in x. Then, for $V = \{a_1, \ldots, a_n\}$, the *Parikh mapping* associated with V is the mapping on V^* defined by $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$ for each $x \in V^*$. The family of recursively enumerable languages is denoted by RE, and the Parikh images of languages in RE is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers). The family of all recursively enumerable sets of natural numbers is denoted by nRE.

The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string. Clearly, using strings is only one of many ways to specify multisets.

We define now a membrane system, which in addition to the most basic features discussed in the Introduction uses also *priority* relations on evolution rules, and the membrane *dissolving* capability.

Such a membrane system is called a P *system*, and it is a construct

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

where:

(i) V is an alphabet – its elements are called *objects*;

(ii) $T \subseteq V$ (the *output* alphabet);

(iii) $C \subseteq V - T$ (*catalysts*);

(iv) μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) injectively labeled by the elements of a given set H of m labels (in this paper $H = \{1, 2, \ldots, m\}$); m is called the *degree* of Π;

(v) $w_i, 1 \leq i \leq m$, are strings which represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ;

(vi) An *evolution rule* is a pair (u, v), which we will usually write in the form $u \rightarrow v$, where u is a string over V and $v = v'$ or $v = v'\delta$, where v' is a string over $\{a_{here},\ a_{out},\ a_{in_j} \mid a \in V, 1 \leq j \leq m\}$, and δ is a special symbol not in V. The length of u is called *the radius* of the rule $u \rightarrow v$.
$R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over V – each R_i is associated with the region i of μ; ρ_i is a partial order relation over R_i, called a *priority* relation (on the rules of R_i).

To simplify the notation, the subscript "here" for letters (objects) in evolution rules will be mostly omitted.

If Π contains rules of radius greater than one, then we say that Π is a system *with cooperation*. Otherwise, it is a *non-cooperative* system. A particular class of cooperative systems is that of *catalytic* systems: the only rules of a radius greater than one are of the

form $ca \to cv$ or $ca \to cv\delta$, where $c \in C, a \in V - C$, and $v \in (V - C)^*$; moreover, no other evolution rules contain catalysts (i.e., there are no rules of the form $c \to v$ or $a \to v_1 c v_2$, with $c \in C$ and $a \in V - C$).

Now since we have a priority relation and the dissolving capability, we have to modify the description of the single macro-step of a computation in a membrane system given in the Introduction.

To take into account the dissolving action δ we add as the last step the following micro-step: for each region where a rule containing δ was used, the membrane enclosing this region is removed, and consequently the objects of this region will belong now to the region that was enclosing the dissolved membrane. Obviously, if the membrane of this region was also dissolved, then the objects "travel" even further up. Since the skin membrane is never dissolved, there is a limit to this travel. Note that the evolution rules in each region are associated with this region, and so, if the region disappears because the membrane enclosing the region is dissolved, then the associated evolution rules also disappear.

To take care of the priority relation we modify the first micro-step as follows: an object can be assigned to a rule only if no object can be assigned to a rule of a higher priority. Hence, we have a competition for rule application and not a competition for choosing perhaps the same objects.

The $(m+1)$-tuple $(\mu, w_1, \ldots, w_m)$ constitutes the *initial configuration* of Π. Since we have the possibility of dissolving membranes, the system may enter a configuration which will include only some of the initial membranes, Thus, any sequence $(\mu', w'_{i_1}, \ldots, w'_{i_k})$, with μ' a membrane structure obtained by removing from μ all membranes different from $i_1, \ldots, i_k$ (of course, the skin membrane is not removed), with w'_{i_j} strings over V, $1 \leq j \leq k$, and $\{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$, is called a *configuration* of Π. Note that not every configuration may be *reachable* through an evolution of the system. Also, note that if a membrane is present in two different configurations, then it will have the same label, because labels are associated with membranes (and never manipulated during an evolution of the system).

For two configurations $C_1 = (\mu', w'_{i_1}, \ldots, w'_{i_k})$, $C_2 = (\mu'', w''_{j_1}, \ldots, w''_{j_l})$ of Π we write $C_1 \Longrightarrow C_2$, and we say that we have a *transition* from C_1 to C_2, if we can pass from C_1 to C_2 by using the evolution rules from $R_{i_1}, \ldots, R_{i_k}$ in the regions $i_1, \ldots, i_k$.

We emphasize here the fact that when using a rule $u \to v$ in the region i_t, copies of the objects as specified by u are "consumed" (removed), and the result of using the rule is determined by v.

The macro-steps corresponding to the use of evolution rules are performed in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the regions associated with the rules, for all regions, following the principles of nondeterminism and maximal parallelism as discussed in the Introduction.

A sequence of transitions between configurations of a given P system Π is called a *computation* with respect to Π. A computation is *successful* if and only if it halts, that is, there is no rule applicable to the objects present in the last configuration. The result (output) of a successful computation is $\Psi_T(w)$, where w describes the multiset of objects from T sent out of the system during the computation (a non-successful computation has no output). The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$ ("Ps" stands for "Parikh

set"), and we say that it is *generated* by Π.

We illustrate the above definitions with the following **example**. Consider the following P system (of degree 3):

$$\begin{aligned}
&\Pi = (V, T, C, \mu, w_1, w_2, w_3, (R_1, \rho_1), (R_2, \rho_2), (R_3, \rho_3)),\\
&V = \{a, b, d, e, f\},\ T = \{e\},\ C = \emptyset,\\
&\mu = [_1[_2[_3\ ]_3]_2]_1,\\
&w_1 = \lambda,\ R_1 = \{e \to e_{out}\},\ \rho_1 = \emptyset,\\
&w_2 = \lambda,\ R_2 = \{b \to d,\ d \to de,\ r_1 : ff \to f,\ r_2 : f \to \delta\},\ \rho_2 = \{r_1 > r_2\},\\
&w_3 = af,\ R_3 = \{a \to ab,\ a \to b\delta,\ f \to ff\},\ \rho_3 = \emptyset.
\end{aligned}$$

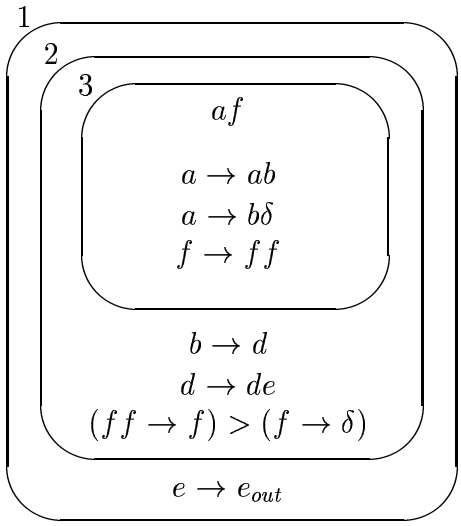

Figure 3: The initial configuration of Π (with rules included)

The initial configuration of Π (including the rules) is given in Figure 3. No objects are present in regions 1 and 2, and so no rules can be applied in these regions. Hence one has to start in region 3, using the single copies of objects a and f. If we iterate the use of rules $a \to ab$ and $f \to ff$, in parallel for all occurrences of a and f currently available, then after n steps, $n \geq 0$, we get n occurrences of b and 2^n occurrences of f. If we then use $a \to b\delta$ instead of $a \to ab$ (note that we always have only one copy of a), then we get $n+1$ occurrences of b and 2^{n+1} occurrences of f, and, moreover, we dissolve membrane 3 (and so region 3 disappears). This means that all the occurrences of objects from region 3 become occurrences of objects from region 2, the rules of region 3 are "lost" (removed), and the rules of region 2 can now be applied to all occurrences of objects present in region 2. As dictated by the priority relation, we have to use the rule $ff \to f$ as much as possible. In one step, we transform b^{n+1} to d^{n+1}, while the number of occurrences of f is halved. In the next step, $n+1$ occurrences of e are produced: each occurrence of d introduces one occurrence of e. At the same time, the number of occurrences of f is halved again.

The priority relation ensures that this step must be iterated n times (each time producing $n+1$ occurrences of e), and then the rule $f \to \delta$ must be used. Its use dissolves membrane 2 (and so the rules of region 2 are removed), while the objects of region 2 become objects of the skin region, which contains only rules for e. Now, in one step, all copies of this object will be sent out of the system, by using the rule $e \to e_{out}$. No further step is possible, and so the computation stops.

Figure 4 illustrates a halting computation in Π, for $n = 4$.

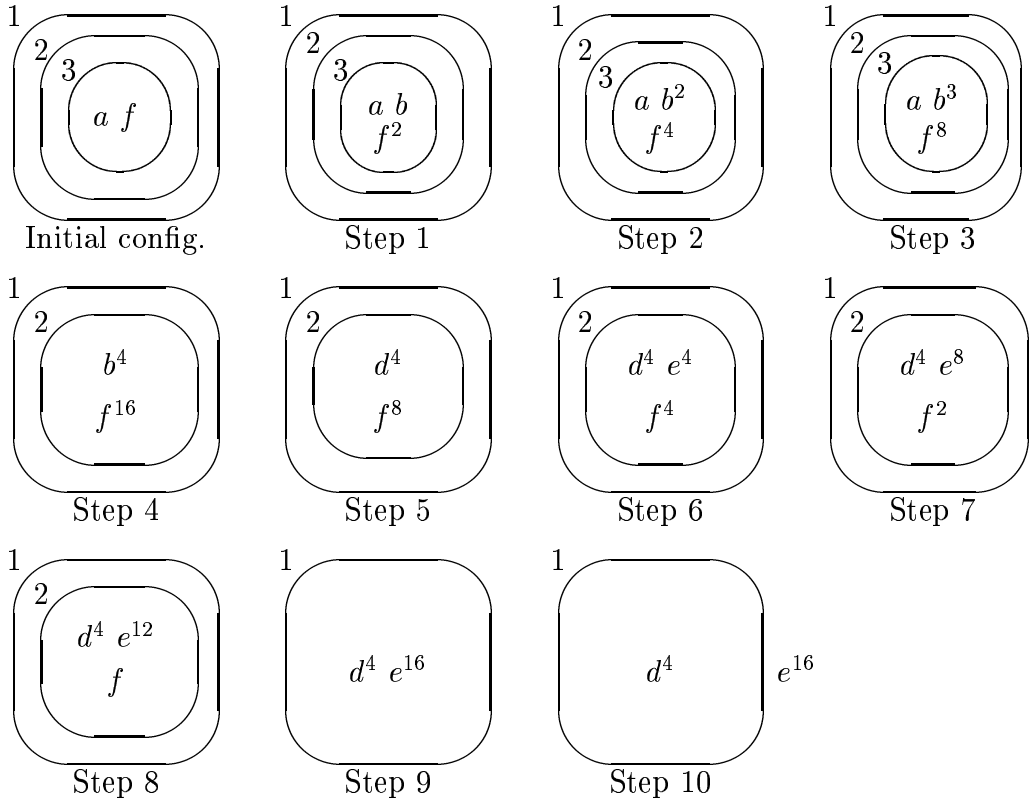

Figure 4. A computation in the P system Π from the example

Hence, for this computation the result is $4^2 = 16$. More general, we send out of the system $(n+1)(n+1)$ copies of the object e, for some $n \geq 0$. Hence $Ps(\Pi) = \{(n^2) \mid n \geq 1\}$.

4 Organizing Communication

Communication between regions plays a central role in computations in P systems. Such a communication consists of exchanging objects between regions, where the exchange is programmed by the use of addressing (subscripts of objects) *here, out*, and in_j which precisely indicates the region destination of objects introduced by evolution rules. In particular, in_j is a very powerful form of addressing: an object with this subscript introduced in region i will enter region j only if region j is directly contained in region i; otherwise

an evolution rule using in_j addressing cannot be used. This "adjacency checking" is a powerful feature of programming computations in P systems.

There are a number of ways of weakening the programming power provided by in_j.

The obvious way is to replace in_j addressing by in addressing: if an object a_{in} is introduced in region i, then a will enter any of the adjacent lower regions, chosen nondeterministicaly among all adjacent lower regions; if membrane i is elementary, then rules using in addressing cannot be used.

The addressing using in_j associates a specific object with a specific membrane (hence a region). An alternative way, more specific than in and less specific than in_j, is to associate both with objects and membranes (*electrical*) *charges* of three sorts: $+, -, 0$ (positive, negative, neutral). The charges of membranes are given in the initial configuration, and they are not changed during computations, the charge of objects are given explicitly by the evolution rules that introduce them. For example, the rule $a \to b^+d^-$ introduces one occurrence of b positively charged, and one negatively charged occurrence of d. A charged object will immediately go through one of the directly adjacent lower membrane of the opposite charge, the neutral objects remain in the same region or will exit it, depending on whether they have the subscript *here* or the subscript *out*, respectively. After a charged object crosses a membrane, it becomes neutral.

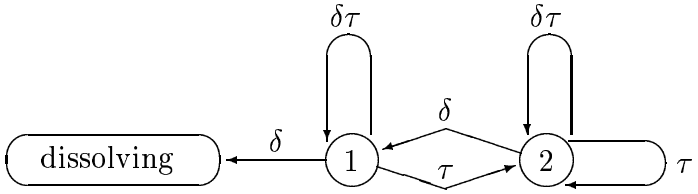

Figure 5: The effect of actions δ, τ

Another way of controlling the passage of objects through membranes is to control the *permeability* of membranes. It is well-known that the permeability of (real life) biomembranes can be variable. This control of permeability is implemented in P systems by the use of action τ, which can increase the "thickness" of a membrane making it impermeable. Together, actions δ and τ provide a very convenient programming device to regulate the passage of objects through membranes. Let us asssume that initially all membranes have thickness 1. If a rule within a membrane of thickness 1 introduces the symbol τ, then the thickness of this membrane becomes 2. A membrane of thickness 2 does not become thicker through the use of other rules which introduce the symbol τ, however no object can pass through it. If a rule which introduces the symbol δ is used within a membrane of thickness 1, then the membrane is dissolved; if the membrane had thickness 2, then it returns to thickness 1. If within the same step one uses rules which introduce both δ and τ in the same membrane, then the membrane does not change its thickness. The cumulative effect of the actions δ and τ are illustrated by the diagram in Figure 5. It turns out that the control of the membrane permeability is a very powerful computing tool (see Section 5).

5 Universality

In this section we consider the computational power of our basic model, equipped with the features discussed above.

The family of sets of vectors over natural numbers $Ps(\Pi)$ generated by P systems of degree at most $m \geq 1$, with priority and catalysts, using target indications of the form *here, out, in*$_j$, and also using the actions δ, τ, is denoted by $NP_m(Pri, Cat, tar, \delta, \tau)$; whenever one of the features $\alpha \in \{Pri,\ Cat,\ \delta, \tau\}$ is not used, we replace it with $n\alpha$. Hence, e.g., $NP_m(nPri, Cat, tar, n\delta, n\tau)$ denotes the family of sets of vectors over natural numbers generated by the most basic membrane systems discussed in the Introduction.

When we use the communication commands *here, out, in*, then we replace *tar* with i/o; when we use electrical charges, we write $\pm$ instead of *tar*.

The following theorem (based on the main results from [40], [16]) is the basic universality result for P systems.

Theorem 5.1 $PsRE = NP_2(Pri, Cat, i/o, n\delta, n\tau) = NP_4(nPri, Cat, i/o, \delta, \tau)$.

The proofs of these equalities are based on a technique used in the proofs of many universality results for P systems. It can be briefly explained (for the reader familiar with basic language theory) as follows. It is known that the recursively enumerable languages are also generated by *matrix grammars with appearance checking* (ac), a class of regulated context-free grammars already well investigated in the sixties, see [53], [13]. Moreover, a *binary normal form* is valid for such grammars (see Lemma 1.3.7 in [13]). The number of nonterminals which appear in rules which are used in the ac mode can be bounded (by two – see [17], which has improved the bound six from [39]). Starting from a matrix grammar G with appearance checking, in the binary normal form, and with at most two nonterminals used in rules which can be applied in the ac mode, we can construct a P system Π_G of the type $(nPri, Cat, i/o, \delta, \tau)$ simulating G and with only four membranes: the skin membrane and one inner membrane simulate the matrices without ac rules, while two further membranes can take care of the two nonterminals which appear in matrices which contain ac rules, thus also simulating these matrices – that's why four membranes suffice. If priority relations among rules are also used, then two membranes suffice (even without using the membrane thickness control). In either case, only weak addressing (using commands *here, out, in*) is used.

The use of catalysts is very convenient in programming computations in P systems. Their main role is to keep the parallelism under control: we have to simulate a sequential device (a matrix grammar) in an inherently parallel framework (a P system), hence we have to "inhibit" the parallelism, and this is done by using (a limited number of) catalysts. When these catalysts have a sort of "short term memory", then even one membrane suffices for obtaining universality. Each such catalyst (called *bi-stable*) has two states, c and $\overline{c}$, and they are used in rules of the form $ca \to \overline{c}v$ and $\overline{c}a \to cv$ (always changing from c to $\overline{c}$ and back). We write $2Cat$ instead of Cat in denoting the generated families of vector sets. The proof of the following result (demonstrating the power of bi-stable catalysts) can be found in [49].

Theorem 5.2 $PsRE = NP_1(nPri, 2Cat, i/o, n\delta, n\tau)$.

6 Trading Evolution for Communication

Let us now consider, following [34], a "purely communicative" class of P systems, where the objects never change (never evolve) – they only pass through membranes. Due to the fact that the number of objects present in the system at any time should be increased/decreased in a controlled manner, we use *carriers*, and provide sufficient copies of each object in the environment.

The origin of this idea is twofold. It abstracts the work of the carrier proteins assisting molecules to pass through membranes (see, e.g., [2]), and it also abstracts from the fundamental idea of 'vectors' used in gene cloning (see, e.g., [9]).

Thus, in membrane systems with carriers we have objects of two types: the carriers ("vehicles") and the passengers. None of them ever changes, and moreover the passengers can pass through membranes only when carried by carriers. We also have objects (both carriers and passengers) available in the environment. Rules to handle objects (attaching/detaching carriers to/from passengers, and passing through membranes) are associated with regions, and also with the environment. Otherwise, the functioning of a membrane system with carriers is the same as an ordinary membrane system: rules are applied in a nondeterministic maximally parallel manner, and transitions between configurations yield computations.

It is also worth mentioning that in this case no object is created or destroyed, only the location of the objects can be changed. Hence, the "conservation law" is observed – which does not necessarily happen in other classes of P systems.

The rules used in P systems with carriers are of the following four types (V is the set of vehicle-objects and O is the set of passenger-objects):

- $va_1 \dots a_k \to [va_1 \dots a_k]$, for $v \in V, a_1, \dots, a_k \in O, k \geq 1$ (*attaching rules*);
- $[va_1 \dots a_k] \to va_1 \dots a_k$, for $v \in V, a_1, \dots, a_k \in O, k \geq 1$ (*detaching rules*);
- $[va_1 \dots a_k] \to in$, for $v \in V, a_1, \dots, a_k \in O, k \geq 0$ (*carry-in rules*);
- $[va_1 \dots a_k] \to out$, for $v \in V, a_1, \dots, a_k \in O, k \geq 0$ (*carry-out rules*);

where v is a carrier and $a_1, \dots, a_k$ are passengers. In the environment there are only rules of the first three forms. The maximal number of passengers in any rule of a system is called *the carring index* of the system.

The family of all sets $Ps(\Pi)$ computed by systems with carriers of degree at most $m \geq 1$, using at most $p \geq 1$ carriers, and with the carrying index not exceeding $k \geq 1$, is denoted by $NCP_m(p, k)$; when any of the parameters m, p, k is not limited, we replace it by $*$.

The following results are from [34]. (Note that, again, two membranes suffice, while the carrying index is rather low.)

Theorem 6.1 $PsRE = NCP_2(3, 3) = NCP_2(*, 2)$.

7 Structuring Objects (Strings)

In a cell, many objects can be considered as being *atomic* (with no internal structure), but many other objects, such as, e.g., DNA molecules, have a structure, which, sometimes,

can be described by a string. This leads one to consider P systems where objects are strings.

This points to a general observation/analogy: in Lindenmayer systems (see, e.g., [51], [52]) the basic unit is a *cell* with no internal structure assigned to it; in P systems that we have considered so far, one "zooms" into a cell, distinguishing the membrane structure and the objects contained within its compartments. Now we will zoom one level "deeper": the objects will be structured (as strings). If we focus at this level of abstraction, then we are again within the framework of formal language theory, and when we choose splicing as the basic operation, then we are in the framework of H systems (see, e.g., [19], [44]).

It is natural, when working with string-objects, to use string processing rules as evolution rules. We will consider first the simplest case, when the multiplicity is ignored and we deal with formal languages in the classic sense.

7.1 P Systems with Rewriting

One natural way to process string-objects is to use rules of the form $(X \to v; tar)$, where $X \to v$ is a usual context-free rule and tar is a target indication, one of *here, out, in*, specifying in the standard way the region where the result of rewriting should go. We can also append to v the symbols δ and τ, which control the membrane thickness in the way discussed in Section 4.

The structure and the functioning of a rewriting P system are defined in the usual way, with the following additional observations: all strings are processed in parallel, but each single string is rewritten by only one rule (the parallelism is maximal at the level of strings and rules, but the rewriting is sequential at the level of the symbols from each string). One begins with finite sets of strings in each region, then one applies rewriting rules, and collects the strings over the terminal alphabet which leave the system during a computation – again only halting computations are considered successful. We denote by $RP_m(Pri, i/o, \delta, \tau)$ the family of languages generated in this way by rewriting P systems with at most m membranes, using priorities among the rewriting rules, target indications *here, out, in*, and the actions δ, τ. As usual, we write $n\alpha$ instead of $\alpha \in \{Pri, \delta, \tau\}$ whenever the corresponding feature is not used.

Here is a simple **example**. Consider the rewriting P system

$$\begin{aligned}
\Pi &= (V, T, \mu, M_1, M_2, (R_1, \rho_1), (R_2, \rho_2)),\\
V &= \{a, b, c, d, d', e, e'\},\ T = \{a, b, c\},\\
\mu &= [_1[_2\ ]_2]_1,\\
M_1 &= \{de\},\ M_2 = \emptyset,\\
R_1 &= \{(d \to ad'b, here),\ (e \to ce', in),\ (d \to ab, here),\ (e \to c, out)\},\ \rho_1 = \emptyset,\\
R_2 &= \{(d' \to d, out),\ (e' \to e, here)\},\ \rho_2 = \emptyset.
\end{aligned}$$

Assume that we have a string of the form $a^n db^n c^m e$ in the skin region, with $n, m \geq 0$ (initially we have $n = m = 0$). If we apply the rule $(e \to ce', in)$, then the string $a^n db^n c^{m+1} e'$ is sent to region 2, we apply then the rule $(e' \to e, here)$, and the computation halts without sending out any string. Thus, in the skin membrane we have to use both rules $(d \to ad'b, here)$ and $(e \to ce', in)$. The string $a^{n+1} d' b^{n+1} c^{m+1} e'$ is sent to region

2. If we use here only the rule $(d' \to d, out)$, then the string $a^{n+1}db^{n+1}c^{m+1}e'$ is sent to region 1, where we can use one of the rules $(d \to ad'b, here)$ or $(d \to ab, here)$, but the computation halts again without producing any output. We have to use both rules from region 2, hence we return to the skin membrane the string $a^{n+1}db^{n+1}c^{m+1}e$. The process can be iterated, resulting in the simultaneous increase of the number of occurrences of symbols a, b, c. If in the skin region we use the rules $(d \to ab, here)$ and $(e \to ce', in)$, then again we cannot exit from the inner membrane. Also, if we send out of the system a string by using the rule $(e \to c, out)$ before using the rule $(c \to ab, here)$, then the string is not accepted because it contains the symbol d, which is not in T. In this way we generate the non-context-free language $L(\Pi) = \{a^n b^n c^n \mid n \geq 1\}$.

The following result is from [23], [59].

Theorem 7.1 $RE = RP_2(Pri, i/o, n\delta, n\tau)$.

The use of a priority relation can be avoided at the cost of controlling the membrane thickness. This was first considered in [59], [60], where a characterization of RE has been given, using systems without a bound on the number of membranes. The following result is from [16].

Theorem 7.2 $RE = RP_4(nPri, i/o, \delta, \tau)$.

One can combine the rewriting of strings with their duplication. A rewriting-replication rule (see [24]) is of the form $r : X \to (u_1, tar_1)|| \ldots ||(u_n, tar_n)$. To apply r to a string w one replaces one occurrence of X in w by u_1, by u_2, and so on, in a context-free manner (i.e., X is replaced and the rest of w is just replicated). Thus, this rewriting yields n strings, $w_1u_1w_2, \ldots, w_1u_nw_2$, where $w = w_1Xw_2$. As usual, these n strings are sent to regions as indicated by the targets $tar_1, \ldots, tar_n$, respectively.

The family of all languages $L(\Pi)$, generated by rewriting-replicating systems Π of degree at most $m \geq 1$ is denoted by $RRP_m(i/o)$ (no further feature is used, such as Pri, δ, τ). The following characterization of RE is from [28].

Theorem 7.3 $RE = RRP_6(i/o)$.

7.2 P Systems with Splicing

An attractive variant is to process the string-objects by the splicing operation, introduced in [19] as a formal model of DNA recombination under the influence of restriction enzymes and ligases (see [44] for a comprehensive investigation of splicing systems).

Consider an alphabet V and two symbols $\#, \$$ not in V. A *splicing rule* over V is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$. For such a rule r and for $x, y, w, z \in V^*$ we define

$$(x, y) \vdash_r (w, z) \quad \text{iff} \quad x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ w = x_1u_1u_4y_2,\ z = y_1u_3u_2x_2,$$
$$\text{for some } x_1, x_2, y_1, y_2 \in V^*.$$

(One cuts the strings x, y in between u_1, u_2 and u_3, u_4, respectively, and then recombines the fragments obtained in this way.)

In a *splicing P system* Π, the rules are given in the form $(r; tar_1, tar_2)$, where $r = u_1\#u_2\$u_3\#u_4$ is a splicing rule over V, and $tar_1, tar_2 \in \{here, out, in\}$.

As usual in splicing systems, when a string is present (in a region of Π), it is assumed to appear in arbitrarily many copies.

The transitions among configurations of a splicing P system are defined by applying the splicing rules from each region, in parallel, to all possible strings from the region, and following the target indications associated with the rules. More specifically, if x, y are from region i and $(r = u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$ is a rule from region i such that we can have $(x, y) \vdash_r (w, z)$, then w and z will go to the regions indicated by tar_1, tar_2, respectively. Note that the strings x, y are still available in region i, because we have assumed that they appear in arbitrarily many copies (an arbitrarily large number of them were spliced, arbitrarily many remain). However, if a string w, z obtained by splicing is sent out of region i, then no copy of it remains in region i.

The result of a computation consists of all strings which are sent out of the system at any time during the computation (here we do not work with halting computations, because the computations which contain at least one transition never halt, due to the assumption that the strings are not consumed by splicing). We denote by $SP_m(i/o)$ the family of languages generated in this way by splicing P systems of degree at most $m, m \geq 1$.

Splicing P systems with three membranes either arranged in two levels or in three levels, were shown in [48] to characterize the recursively enumerable languages. The following result from [38] shows that two membranes suffice.

Theorem 7.4 $RE = SP_2(i/o)$.

7.3 Counting the Copies of String-Objects

We will consider now P systems handling *multisets of strings*, and where the result of a computation is the number of strings sent out of the system during a halting computation, and not the strings themselves. The number of copies of strings is important, so therefore we need to consider operations on strings which can increase this number (we do not also need operations which decrease the number of strings, because we can achieve this by storing strings in certain membranes, which can act as "garbage collectors").

The operations that we will use are *replication*, as defined at the end of Subsection 7.1, and *splitting*. If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1|u_2$ is called a *splitting rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Longrightarrow_r (w_2, w_3)$ (and we say that w_1 is split by rule r) if $w_1 = x_1ax_2,\ w_2 = x_1u_1,\ w_3 = u_2x_2$, for some $x_1, x_2 \in V^*$.

P systems with multisets of string-objects, processed by rewriting, replication (into two new strings), splitting, and *recombination/crossover* (for $z, w_1, w_2, w_3, w_4 \in V^+$, we write $(w_1, w_2) \Longrightarrow_z (w_3, w_4)$ if $w_1 = x_1zx_2,\ w_2 = y_1zy_2$, and $w_3 = x_1zy_2,\ w_4 = y_1zx_2$, for some $x_1, x_2, y_1, y_2 \in V^*$) were considered in [11].

These rules have associated targets *here, out, in* for the resulting strings (two in the case of replication, splitting, and recombination, and one in the case of rewriting), and they are applied as usual in P systems. A string which enters an operation is "consumed" by that operation, its multiplicity is decreased by one. The multiplicity of strings produced

by an operation is increased accordingly. A string is processed by one operation only. For instance, we cannot apply two rewriting rules, or a rewriting rule and a replication rule, to the same string.

The result of a halting computation consists of the number of strings sent out of the system during the computation. We denote by $NWP_m(i/o)$ the sets of numbers computed by all P systems of this type, using *in, out, here* addressings (in [11] they were called *P systems with worm-objects*, following the terminology of [54], where similar operations were used) with at most $m \geq 1$ membranes.

The following result is from [32].

Theorem 7.5 $nRE = NWP_6(i/o)$.

Going further, we can now consider systems with a combination of operations. An example of such a system is a P system with worm-objects, taking as the result of a computation the strings themselves sent out of the system rather than their number (that is, we work with multisets of strings, but we generate languages, not sets of numbers). The case when rewriting and crossover operations are used was considered in [33]. We denote by $RXP_m(i/o)$ the family of languages generated by P systems with at most $m \geq 1$ membranes using these operations. The proof of the following result can be found in [33]

Theorem 7.6 $RE = RXP_5(i/o)$.

8 Trading Time for Space

We now address the important issue of the computing efficiency of P systems.

An important theorem from [58] says that each deterministic P system of the type $(Cat, Pri, tar, \delta, \tau)$ (hence working with symbol-objects, and using all features: catalysts, priorities, the control of membrane thickness, and addressing by in_j) can be simulated by a deterministic Turing machine with a polynomial slowdown. This means that by using such systems we cannot solve exponential problems in polynomial time, in spite of the fact that exponentially many objects can be produced in linear time, for instance, by rules of the form $a \to aa$. Therefore, in order to improve the computational performance of our systems it is necessary to provide more efficient ways for producing an exponential space. Three such ways will be discussed in the following subsections.

8.1 Dividing Membranes

One possibility to get exponential space is to consider *membrane division* (and this also corresponds to a common biological phenomenon). This feature was considered in [43] for *systems with active membranes*, where the membranes themselves are involved in rules. We recall the definition in a restricted form (considered in [37]).

A *P system with active membranes*, is a construct $\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$, where:

(i) $m \geq 1$;

(ii) V is the alphabet of the system;

(iii) H is a finite set of *labels* for membranes;

(iv) μ is a *membrane structure*, consisting of m membranes labeled with elements of H and having a neutral charge (initially, all membranes are neutrally charged);

(v) $w_1, \ldots, w_m$ are strings over V, describing the *multisets of objects* placed in the m regions of μ;

(vi) R is a finite set of *rules*, of the following forms:

(a) $[_h a \to v]_h^\alpha$, for $h \in H,\ a \in V,\ v \in V^*,\ \alpha \in \{+, -, 0\}$ (object evolution rules),

(b) $a[_h]_h^\alpha \to [_h b]_h^\beta$, where $a, b \in V,\ h \in H,\ \alpha, \beta \in \{+, -, 0\}$ (an object is introduced in membrane h),

(c) $[_h a]_h^\alpha \to [_h]_h^\beta b$, for $h \in H,\ \alpha, \beta \in \{+, -, 0\},\ a, b \in V$ (an object is sent out of membrane h),

(d) $[_h a]_h^\alpha \to b$, for $h \in H,\ \alpha \in \{+, -, 0\},\ a, b \in V$ (membrane h is dissolved),

(e) $[_h a\ ]_h^{\alpha_1} \to [_h b\ ]_h^{\alpha_2} [_h c\ ]_h^{\alpha_3}$, for $h \in H$ an elementary membrane, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in V$
(2-division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, maybe of different polarity; the object specified in the rule is replaced in the two new membranes by possibly new objects; all other objects are duplicated in the two new copies of the membrane).

The rules are used as usual in a P system, in a maximally parallel manner: in each time unit, all objects which can evolve, have to evolve. Each copy of an object and each copy of a membrane can be used by only one rule, with the exception of rules of type (a), where we count only the involved object, not also the membrane. That is, if we have several objects a in a membrane i and a rule $[_i a \to v]_i^\alpha$, then we use this rule for all copies of a, irrespective of how many there are; we do not consider that the membrane was used – note that its electrical charge is not changed. However, if we have a rule $[_i a]_i^\alpha \to [_i\ ]_i^\beta b$, then this counts as using the membrane, no other rule of types (b), (c), (d), (e) which uses the same membrane can be applied at the same time. When dissolving a membrane, its contents becomes a part of the contents of the directly surrounding membrane; when dividing a membrane, its contents are replicated in the two obtained membranes. Only elementary membranes can be divided. The skin membrane can neither dissolve nor divide, but it can be "electrically charged". During a computation, objects can leave the system (by using rules of type (c)).

A natural extension is to allow also the division of non-elementary membranes by rules of the form $[_h a]_h^{\alpha_1} \to [_h b]_h^{\alpha_2} [_h c]_h^{\alpha_3}$; in such a case, not only the objects from the former membrane, but also the membranes present in it are replicated in the two newly obtained membranes. Such a rule is said to be of type (e′).

The family of all sets of vectors $Ps(\Pi)$, computed by systems which use simultaneously at most n membranes, and using rules of the forms (a),...,(e) is denoted by $NAP_n((a), \ldots, (e))$; (e) is replaced by (e′) if non-elementary membranes can be divided. When a type of rule is not used, we remove the corresponding index from the list (a),...,(e)/(e′).

The proof of the following result can be found in [16].

Theorem 8.1 $PsRE = NAP_4((a),(b),(c))$.

More significant than this (expected) result is the following one, from [37].

Theorem 8.2 *The Hamiltonian Path Problem (HPP) can be solved in quadratic time and the SAT problem can be solved in linear time by P systems with active membranes, using rules of the forms (a),...,(d), (e').*

Similar results are given in [43], but also using the possibility of dividing a membrane under the influence of inner membranes, not only under the influence of an object, as in rules of types (e) and (e'). In [22] one considers the possibility of dividing a membrane in an arbitrary number of copies (not only two), and solutions to the HPP and the Node Covering Problem in that framework were proposed. The proofs are based on constructing a P system associated with a graph and generating all paths from a specified initial node to a specified final node, then checking whether or not at least one of these paths is Hamiltonian. This directly corresponds to the way HPP is solved in [1], but here the generation of all paths takes a quadratic time, because both the number of nodes and the maximal outdegree of the graph count. As in [25], we can then reduce SAT to a problem of paths in a graph with each node having the outdegree two, and so we obtain a linear time solution to this problem.

8.2 Replicating Strings

Another powerful way to obtain exponential space sufficient for solving NP-complete problems in polynomial time is to use the replication of string-objects, as considered in P systems with replicated rewriting (as in Theorem 7.3) and in systems with worm-objects. It was proved in [11] and [24] that HPP and SAT can be solved in linear time by such systems. (If the replication produces only two new strings, then HPP reauires a quadratic time – see [11].)

The replication of strings can be obtained not only in a "direct" way, by replicating rules as mentioned above, but also in an "indirect" manner, starting from a conditional way of communicating objects through membranes. The basic idea is to consider certain predicates on strings and communication rules of the form $(\pi, in_j), (\pi, out)$, with the meaning that if $\pi(w) = true$, then the string w must follow the addressings in_j, out. A variant is to send the string w to one of these targets, nondeterministically choosing it, but we may also choose to send the string *to all* membranes for which a predicate holds true. That is, we replicate the string in as many copies as many communication predicates are true.

Predicates for controlling the string-object communication were considered in [8], but without investigating the computational efficiency of the replication. This was done in [31], for the so-called *P systems with valuations*, introduced in [30]: a morphism from symbols to integer numbers assigns "valuations" to strings; the sign of this valuation is interpreted as an electrical charge and used for communicating the string as discussed in Section 4 (a string of a given polarization goes to a membrane of the opposite polarization, while the neutral strings remain in the same membrane). When a string can go to several adjacent membranes (for instance, it has polarity + and there are several adjacent

membranes with polarity $-$), then the string is replicated and copies of it are sent to all targets. As expected, by using this idea, polynomial solutions of NP-complete problems can be devised; this is illustrated in [31] by SAT and HPP.

8.3 Creating Membranes

An interesting way for obtaining exponential space is by using the possibility of creating new membranes. For instance, rules of the form $a \to [_i b]_i$ can be used, where a and b are symbol-objects and i is the label of a membrane, from a given list of possible membranes (this is important, because by knowing the label, we know the rules to be applied in the associated membrane). Such rules for creating membranes were considered in [21], where a characterization of Parikh images of ET0L languages is obtained in this way, but they can also be used for producing an exponential space for computations. We illustrate this with the HPP, giving full details, in order to let the reader see an example of a "P algorithm".

Consider a graph $g = (N, E)$ with the nodes $N = \{a_1, a_2, \ldots, a_n\}$. In order to decide whether a Hamiltonian path exists which starts in a_1 and ends in a_n we construct the P system Π with the membrane structure $\mu = [_0[_1\ ]_1]_0$ (the skin membrane is labeled by 0, and it contains a unique membrane, with label 1), with the object $(a_1, 1)$ present in membrane 1, using the following alphabet of objects

$$V = \{(a_i, j), (a'_i, j) \mid 1 \leq i, j \leq n\} \cup \{M \subseteq N \mid M \neq \emptyset\}$$

(notice that the subsets of N are interpreted as symbol-objects); the possible membranes are labeled by $0, 1, 2, \ldots, n-1$, and the associated sets of rules are as follows:

$$\begin{aligned}
R_0 &= \{N \to yes_{out}\}, \\
R_i &= \{(a_i, j) \to (a'_{k_1}, j+1) \ldots (a'_{k_{s_i}}, j+1) \mid (a_i, a_{k_r}) \in E, \text{ for all} \\
&\qquad 1 \leq r \leq s_i, s_i \geq 1, \text{ and } 1 \leq j \leq n-1\} \\
&\cup \{(a'_k, j) \to [_k (a_k, j)]_k \mid 1 \leq k, j \leq n-1\} \\
&\cup \{(a'_n, n) \to \{a_n\}\} \\
&\cup \{M \to (M \cup \{a_i\})_{out} \mid M \subseteq N\}, \text{ for all } i = 1, 2, \ldots, n-1.
\end{aligned}$$

The idea behind this construction is the following. The tuple symbols (a_i, j) encode the fact that we have reached node a_i on a path starting in a_1 which has already passed through j nodes. Each object (a_i, j) introduces as many objects of the form $(a'_k, j+1)$ as many successors of a_i exist in the graph. Then, each object $(a'_k, j+1)$ creates a membrane with label k. That is, the paths we create are encoded in the membrane structure (all the paths in the graph g consisting of at most n nodes are "recorded" as paths from the root to the leaf nodes of the tree describing the membrane structure of Π). When we reach the node a_n or the paths already contain n nodes, this process (it takes $2(n-1)-1$ steps) is finished, and we pass to the second phase of the computation, that of checking whether or not among the generated paths there is one which is Hamiltonian. This process can start only from object (a'_n, n), that is, only if we have reached node a_n after passing through exactly n nodes. After producing an object of the form of a subset of N (at the first

step, this is $\{a_n\}$), we exit the membranes, one by one; when we exit membrane i we add the node a_i to the current set of nodes. In this way, after at most n steps (one for passing from (a'_n, n) to $\{a_n\}$, and $n-1$ for other nodes), we reach the skin membrane with several objects of the form $M \subseteq N$. Only N can exit the skin membrane, sending out the message *yes*, that is, we have an output (after $3n-2$ steps) if and only if the graph g contains a Hamiltonian path from a_1 to a_n.

The results from this subsection and from Subsections 8.2, 8.3 have a special significance in view of the theorem from [58] cited above: when we have exponentially many symbol-objects placed in a bounded number of membranes we can simulate the system by a Turing machine of a similar efficiency (with a polynomial slowdown); when one uses an exponential number of string-objects placed in a bounded number of membranes, or an exponential number of objects placed in an exponential number of membranes this is no longer true. We can "explain" these results by the much greater quantity of information stored in a string or in a membrane structure than in a multiset of symbol objects.

9 Discussion

In this paper we have outlined the basic framework for membrane computing. We have introduced, and discussed a number of programming features and we have given a number of results that are representative for membrane computing. The research in this area is very active now. The current bibliography (February 2000) includes about 90 papers, with some of them motivated by the properties of bio-membranes, and some of them motivated by formal properties of P systems. We will mention now, in a telegraphic style, *some* interesting notions and results not discussed above.

For instance, two natural operations are: to *merge* membranes and to *move* membranes through other membranes. The first operation can be done by rules of the form $[_i\,]_i[_j\,]_j \to [_k\,]_k$; the effect of such a rule is that the contents of membranes i and j are put together in a new membrane, with the label k. Knowing the label k, we also know the rules which can be applied in the region of membrane k. The merge operation was first considered in [20], where several DNA computing experiments done by T. Head and his collaborators were interpreted as membrane computations. Actually, in [20] several other operations on membranes were considered: divide, create, separate (the string-objects which satisfy a given property are encapsulated in a new membrane, created inside the membrane currently in use). More about such operations can be found in [29].

The operation of moving a membrane, together with its contents, through another membrane, corresponds to the biological operations of endocytosis and fagocytosis, and it was considered in [3] in the framework of P systems with active membranes, as a way of avoiding the division of membranes under the influence of inner membranes (this is a type of rule which we have not considered in Subsection 8.1, but it is considered in [43], [22] and in other papers).

One of the most important ways of moving chemicals through bio-membranes is based on concentration differences between adjacent regions. This idea was formalized in [14], but only results for the case of using bi-stable catalysts were given.

Another variant which has a good motivation is that of systems *with energy accounting*:

integer numbers are associated with rules, expressing "quanta of energy" produced or consumed when applying these rules; at each step, we can use a combination of rules only if the total energy in each region is positive; one starts with zero units of energy in the system, and the energy remaining from an application of a step is passed to the next step; however, if the total energy within a membrane is bigger than the threshold associated (in advance) with the membrane, then the membrane is dissolved (and all its energy consumed in this way). Again, characterizations of $PsRE$ are obtained, either with a small number of membranes or a small total quantity of energy present in the system at any time, [47]; it seems that a trade-off between these two parameters holds.

In the systems discussed in this paper we have considered membrane structures which correspond to trees. An attractive generalization is to consider arbitrary graphs (in such a case, the "regions" associated with the nodes do not necessarily have a spatial counterpart in the form of a membrane structure – unless we consider direct communication among regions, corresponding to the inter-celular communication through common protein channels, see [2], [26]). Such variants were considered in [46], where characterizations of $PsRE$ were given through P systems using planar graphs with one-way communication among regions.

In [15] and in a series of subsequent papers, one considers P systems with a sequential use of rules; this makes the use of catalysts unnecessary, hence "purely non-cooperative" characterizations of recursively enumerable sets of numbers are obtained.

An important direction of research concerns the normal forms, mainly with respect to the membrane structure: it is expected that certain structures are easier to handle than others; in particular, the lyposomes are more frequently arranged in a "linear way" (the tree describing the membrane structure is a line) than in a "branched way". Normal form theorems concerning the membrane structure can be found in [46], [50]. Normal forms with respect to the number and the form of rules present in each membrane (for the case of rewriting P systems) were given in [59].

Several papers have considered the fundamental topic of implementing P systems on an electronic computer, either on the existing media or on a purposely designed architecture. The first type of approach appears in [5], [12], [27], [56], while the second one is dealt with in [57]. A definite assessment of these attempts is premature, but up to now, no result of practical significance was obtained.

This leads to considerations concerning the significance of P systems (for biology, for mathematics, and for computing). The approach is clearly motivated from a mathematical point of view, not only because it is natural to (try to) model the cell computational behavior (with a possible future relevance for biology), but also because the new computing model has several intrinsically interesting features. Examples of such features are: the use of multisets, the inherent parallelism, the possibility of devising computations which can solve exponential problems in polynomial time (by making use of an exponential space created in a natural manner). All these features are only potentially useful from a practical computational point of view (a more optimistic comment can be made on using the basic ingredients of P systems in other computational frameworks, such as Artificial Life – see, e.g., [55]). How should one approach the implementation problem? Should one try to develop, in lab, *wet membrane computers* (as this happens now in DNA computing), or it is wiser to try to implement P systems on electronic computers? The

latter approach has a long and quite successful tradition in Natural Computing: Neural Networks and Evolutionary Programming are also biologically inspired, and they led to new computing paradigms of a definite practical use when implemented on the traditional electronic computer. Maybe, this will also be the case for membrane computing, possibly implemented on a devoted architecture, specially designed for P systems.

Anyway, it is clear already now that the idea of computing with membranes is fruitful from a mathematical point of view. The long list of notions and results discussed in this paper is still growing, and, perhaps more importantly, it is accompanied by a long list of open problems and topics for further research. A list of research topics was given in [42], many more others can be found on the web page (the address is given in the first page of this paper). It is perhaps important to realize that the research in P systems is driven by three seemingly contradictory goals: to get variants of P systems as *realistic* as possible (from a biochemical point of view, or from a possible implementation point of view), as *powerful* as possible, and as *efficient* as possible. Above all these, the models should be (mathematically and computationally) as *elegant* as possible. It is our expectation that the field will flourish, hopefully reaching some of these goals.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (November 1994), 1021–1024.

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

[3] A. Atanasiu, C. Martin-Vide, Recursive calculus with membranes, submitted, 2000.

[4] J.P. Banâtre, A. Coutant, D. Le Metayer, A parallel machine for multiset transformation and its programming style, *Future Generation Computer Systems*, 4 (1988), 133–144.

[5] A.V. Baranda, J. Castellanos, R. Molina, F. Arroyo, L.F. Mingo, Data structures for implementing transition P systems in silico, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 21–34.

[6] G. Berry, G. Boudol, The chemical abstract machine, *Theoretical Computer Sci.*, 96 (1992), 217–248.

[7] P. Bottoni, C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane systems with promoters/inhibitors, submitted, 2000.

[8] P. Bottoni, A. Labella, C. Martin-Vide, Gh. Păun, Rewriting P systems with conditional communication, submitted, 2000.

[9] T.A. Brown, *Gene Cloning. An Introduction*, Chapman & Hall, London, 1996.

[10] C. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor and Francis, London, 2000.

[11] J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, *IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000*, La Coruna, Spain, 64–74.

[12] G. Ciobanu, D. Paraschiv, Membrane software. A P system simulator, manuscript, 2000.

[13] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[14] J. Dassow, Gh. Păun, Concentration controlled P systems, *Acta Cybernetica*, to appear.

[15] R. Freund, Sequential P systems, *Workshop on Multiset Processing*, Curtea de Argeş, Romania, 2000, and *Theorietag 2000; Workshop on New Computing Paradigms* (R. Freund, ed.), TU University Vienna, 2000, 177–183.

[16] R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes: Three more collapsing hierarchies, submitted, 2000.

[17] R. Freund, Gh. Păun, On the number of non-terminals in graph-controlled, programmed, and matrix grammars, *Proc. Conf. Universal Machines and Computations*, Chişinău, 2001 (M. Margenstern, Y. Rogozhin, eds.), Springer-Verlag, Berlin, 2001.

[18] P. Frisco, H.J. Hoogeboom, P. Sant, A direct construction of a universal P system, submitted, 2001.

[19] T. Head, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987), 737–759.

[20] T. Head, Aqueous computations as membrane computations, *Romanian J. of Information Science and Technology*, 4, 1-2 (2001).

[21] M. Ito, C. Martin-Vide, Gh. Păun, A characterization of Parikh sets of ET0L languages in terms of P systems, submitted, 2000.

[22] S.N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), 357–367.

[23] S.N. Krishna, R. Rama, On the power of P systems with sequential and parallel rewriting, *Intern. J. Computer Math.*, 77, 1-2 (2000), 1–14.

[24] S.N. Krishna, R. Rama, P systems with replicated rewriting, *J. Automata, Languages, Combinatorics*, to appear.

[25] R.J. Lipton, Using DNA to solve NP-complete problems, *Science*, 268 (April 1995), 542–545.

[26] W.R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford Univ. Press, New York, Oxford, 1999.

[27] M. Maliţa, Membrane computing in Prolog, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 159–175.

[28] V. Manca, C. Martin-Vide, Gh. Păun, On the power of P systems with replicated rewriting, *J. Automata, Languages, Combinatorics*, to appear.

[29] M. Margenstern, C. Martin-Vide, Gh. Păun, Computing with membranes; Variants with an enhanced membrane handling, *Proc. 7th Intern. Meeting on DNA Based Computers*, Tampa, Florida, USA, 2001.

[30] C. Martin-Vide, V. Mitrana, P systems with valuations, in vol. *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 154–166.

[31] C. Martin-Vide, V. Mitrana, Gh. Păun, On the power of P systems with valuations, submitted, 2001.

[32] C. Martin-Vide, Gh. Păun, Computing with membranes. One more collapsing hierarchy, *Bulletin of the EATCS*, 72 (October 2000), 183–187.

[33] C. Martin-Vide, Gh. Păun, String objects in P systems, *Proc. of Algebraic Systems, Formal Languages and Computations Workshop*, Kyoto, 2000, RIMS Kokyuroku, Kyoto Univ., 2000, 161–169.

[34] C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane systems with carriers, *Theor. Computer Sci.*, to appear.

[35] M. Mutyam, K. Krithivasan, Inter-membrane communication in P systems, *Romanian J. of Information Science and Technology*, 4, 1-2 (2001).

[36] M. Mutyam, K. Krithivasan, P systems with active objects: Universality and efficiency, *Proc. Conf. Universal Machines and Computations*, Chişinău, 2001 (M. Margenstern, Y. Rogozhin, eds.), Springer-Verlag, Berlin, 2001.

[37] A. Păun, On P systems with membrane division, in vol. *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 187–201.

[38] A. Păun, M. Păun, On membrane computing based on splicing, in vol. *Where Mathematics, Computer Science, Linguistics, and Biology Meet* (C. Martin-Vide, V. Mitrana, eds.), Kluwer, Dordrecht, 2001, 409–422.

[39] Gh. Păun, Six nonterminals are enough for generating each r.e. language by a matrix grammar, *Intern. J. Computer Math.*, 15 (1984), 23–37.

[40] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).

[41] Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.

[42] Gh. Păun, Computing with membranes (P systems): Twenty six research topics, *Auckland University, CDMTCS Report* No 119, 2000 (www.cs.auckland.ac.nz/ CDMTCS).

[43] Gh. Păun, P systems with active membranes: Attacking NP complete problems, *J. Automat, Languages and Combinatorics*, 6, 1 (2001), 75–90.

[44] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998.

[45] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266.

[46] Gh. Păun, Y. Sakakibara, T. Yokomori, P systems on graphs of restricted forms, *Publicationes Math. Debrecen*, to appear.

[47] Gh. Păun, Y. Suzuki, H. Tanaka, P Systems with energy accounting, submitted, *Intern. J. Computer Math.*, to appear.

[48] Gh. Păun, T. Yokomori, Membrane computing based on splicing, *Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, June 1999, 213–227.

[49] Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397–410

[50] I. Petre, A normal form for P systems, *Bulletin of the EATCS*, 67 (Febr. 1999), 165–172.

[51] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

[52] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.

[53] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[54] M. Sipper, Studying Artificial Life using a simple, general cellular model, *Artificial Life Journal*, 2, 1 (1995), 1–35.

[55] Y. Suzuki, H. Tanaka, Artificial life and P systems, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 265–285.

[56] Y. Suzuki, H. Tanaka, On a LISP implementation of a class of P systems, *Romanian J. of Information Science and Technology*, 3, 2 (2000), 173–186.

[57] Gh. Ştefan, Membrane computation with cellular automata, on a dedicated hardware, *Workshop on Multiset Processing*, Curtea de Argeş, Romania, August, 2000.

[58] Cl. Zandron, Cl. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, in vol. *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 289–301.

[59] Cl. Zandron, G. Mauri, Cl. Ferretti, Universality and normal forms on membrane systems, *Proc. Intern. Workshop Grammar Systems 2000* (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, July 2000, 61–74.

[60] Cl. Zandron, Cl. Ferretti, G. Mauri, Using membrane features in P systems, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 296–320.