

From Cells to Computers: Computing with Membranes (P Systems)¹

Gheorghe PĂUN²

Institute of Mathematics of the Romanian Academy

PO Box 1 – 764, 70700 București, Romania

E-mail: gpaun@imar.ro

Abstract. The aim of this paper is to introduce to the reader the main ideas of Computing with Membranes, a recent branch of (theoretical) Molecular Computing. In short, in a cell-like system, multisets of objects evolve according to given rules in the compartments defined by a membrane structure and compute natural numbers as the result of halting sequences of transitions. The model is parallel, nondeterministic. Many variants have already been considered and many problems about them were investigated. We present here some of these variants, focusing on two central classes of results: (1) characterizations of the recursively enumerable sets of numbers and (2) possibilities to solve NP-complete problems in polynomial – even linear – time (of course, by making use of an exponential space). The results are given without proofs. An almost complete bibliography of the domain, at the middle of October 2000, is also provided.

Keywords: Natural Computing, Molecular Computing, Membrane Computing, P System, Turing computability, NP-complete problem

1 The Computing Bio-Cell

Assertions about the computing-like activities which take place in alive cells can be found in many places, the reactions by which chemical compounds, energy, and information are handled in the complex structure of a cell being often interpreted as computing processes (see, e.g., Bray, 1995, and the references thereof). The incentive to define P systems (initially, in [P32: Gh. Păun, 1998]³, they were called *super-cell* systems) was the question whether or not these statements are just metaphors, or a formal computing device can be abstracted from the cell functioning. As we will immediately see, the answer is positive.

Three are the fundamental features of alive cells which will be used in our computing machineries: (1) the **membrane structure**, where (2) **multisets** of chemical compounds evolve according to prescribed (3) **rules**. Formulated as a slogan, *life means biochemistry plus membranes*. Here are just two quotations supporting this assertion:

The secret of life, the wellspring of reproduction, is not to be found in the beauty of Watson-Crick pairing, but in the achievement of collective catalytic closure. The roots are deeper than the double helix and are based in chemistry itself.

(S. Kauffman, 1995)

¹A preliminary version of this paper has been presented at the Workshop on Grammar Systems, Bad Ischl, Austria, July 2000.

²Work partially supported by a grant of NATO Science Committee, Spain, 2000–2001.

³The references of the form [P*n*: Author, year] point to papers in the P system bibliography given at the end of the paper.

The ordinary textbook talk of DNA as governing cellular or even organismic behavior is rather misleading. If any entity should be thought of as a governor of cellular activity, this should certainly be the membrane.

All major activities of cells are topologically connected to membranes.

(J. Hoffmeyer, 1998)

Note that a cell is a complex body, with several compartments delimited by membranes of various types, that the chemicals evolving in these compartments are of very diverse forms, from single ions to long DNA molecules, while the membranes themselves are composed of chemicals, hence can interact with the chemicals swimming in the aqueous solutions from the compartments; the whole process is governed by the biochemistry rules, applied more or less nondeterministically, more or less simultaneously in all compartments, in parallel to all “objects”.

These observations are the basis of the “formal computing cell” we are going to consider.

It is important to mention from the very beginning that the P systems are not intended to model the cell behavior, but they abstract from this, in the aim of finding a theoretical computing model mimicking the cell behavior. Whether or not P systems will have any relevance for the biological study of cells, as well as whether or not they will have any practical computing significance are premature questions, an intensive research, in an interdisciplinary team, are necessary in order to have a documented answer.

The paper is organized as follows. Section 2 gives some biochemical information about the structure and the functioning of plasma membrane. Abstracting from this “bio-reality”, Section 3 informally introduces the P systems. Their possible computing significance is discussed in Section 4. Section 5 gives a more formal definition of the basic variants of P systems with symbol-objects; some examples are discussed in Section 6, while Section 7 recalls three central results about the computational universality of these systems. Section 8 passes to P systems with string-objects, processed by rewriting or by splicing operations, while Section 9 introduces P systems with string-objects which are present in multisets and are processed by rules which can change their multiplicity. In Section 10, these last systems are shown to be able to solve the SAT problem in linear time. The last section, 11, briefly presents further results from the already numerous papers published or circulated on Internet; details can be found in the almost complete bibliography of the area, at the middle of October 2000, which closes the paper.

2 The Cell Membrane: Structure and Functions

The many (formal) notions which we will soon consider when defining our computing devices will not only be biology-like, but most of them will correspond in a direct way to notions from the biochemistry of the membranes of alive cells.

A cell has a complex structure, with several compartments delimited inside the main membrane by several inner membranes: the nucleus, the Golgi apparatus, several vesicles, etc. In principle, all these membranes are similar, they are *separators* and *filters*, so that we take here as a prototype the plasma membrane. That is why, having in mind the mathematical model which will follow, we recall here only a few elements concerning the structure and the functioning of the plasma membrane, following Alberts et al., 1998, Loewenstein, 1999, Mader, 1996.

The currently accepted model of the membrane structure is that proposed in 1972 by S. Singer and G. Nicolson and known under the name of *the fluid-mosaic model*. 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.

The phospholipid molecules are composed of two main parts: a polar *head* and a non-polar *tail* – see the pictorial representation from Figure 1(a) and the details from Figure 1(b).

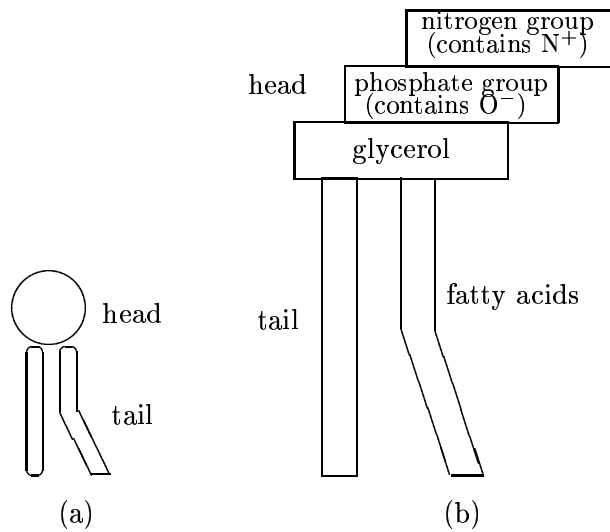


Figure 1: A phospholipid molecule

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. This chemical composition is important for our considerations, because 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 makes easier the exit of negative ions and the entrance of positive ions.

Schematically, a membrane looks like indicated in Figure 2.

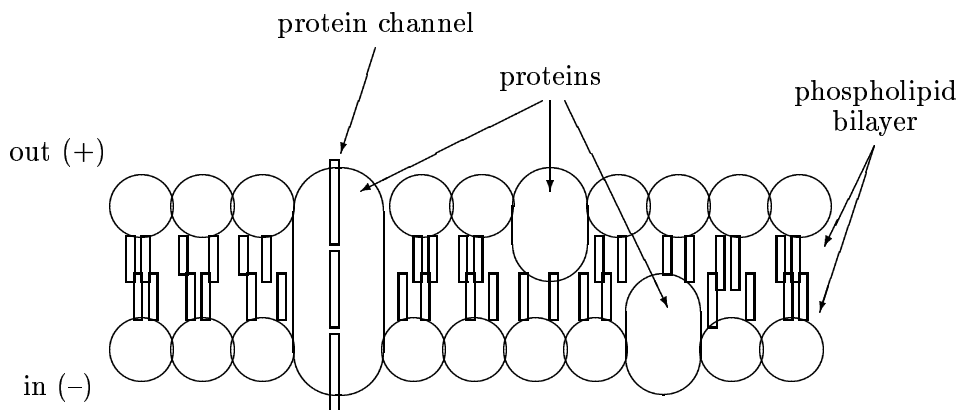


Figure 2: Schematic structure of a membrane

The model is called the fluid-mosaic one, because the phospholipid molecules can move in the two layers, but, because of the heads polarization, always remaining in the planes of the two layers. This makes possible also the move of proteins (and of other compounds), which is

important from the point of view of intra- and inter-cellular communication.

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 cross a membrane only assisted, while charged ions pass selectively from a region to another one.

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

As the reader can already realize, all these details have formal counterparts in the P system area. Another important aspect, also useful for P systems (especially when considering planar membrane-like structures – see, e.g., [P39: Gh. Păun, Sakakibara, Yokomori, 1999]) is the way the neighboring cells establish protein channels for inter-cellular communication: due to the fact that the membrane is a fluid-mosaic, when two membranes touch each other, their proteins can “look for each other”; when two proteins come close enough, they bind and this enhances the formation of further similar protein channels. In this way, a complex communication network can be established through cells (see Figure 3). Interesting enough is that when one of the cells is broken or invaded by “undesired” molecules, its channels to the neighboring cells are closed (inhibited) and they can be re-open later, in favourable circumstances.

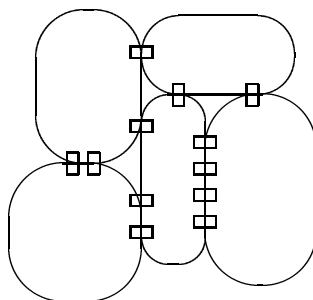


Figure 3: Inter-cellular communication

3 The Computing Math-Cell

Let us now try to abstract from the structure and the functioning of an alive cell and informally describe the symbolic counterpart of the bio-cell. A partial formalization of the notions discussed below can be found in Section 5.

Everything takes place in a *membrane structure*, which is a hierarchical arrangement of membranes, all of them placed in a main membrane, called the *skin* membrane. This one delimits the system from its *environment*. The membranes should be understood as three-dimensional

balloons, but a suggestive pictorial representation is by means of planar Euler-Venn diagrams (see Figure 4). Each membrane precisely identifies a *region*, the space between it and all the directly inner membranes, if any exists. A membrane without any membrane inside is said to be *elementary*.

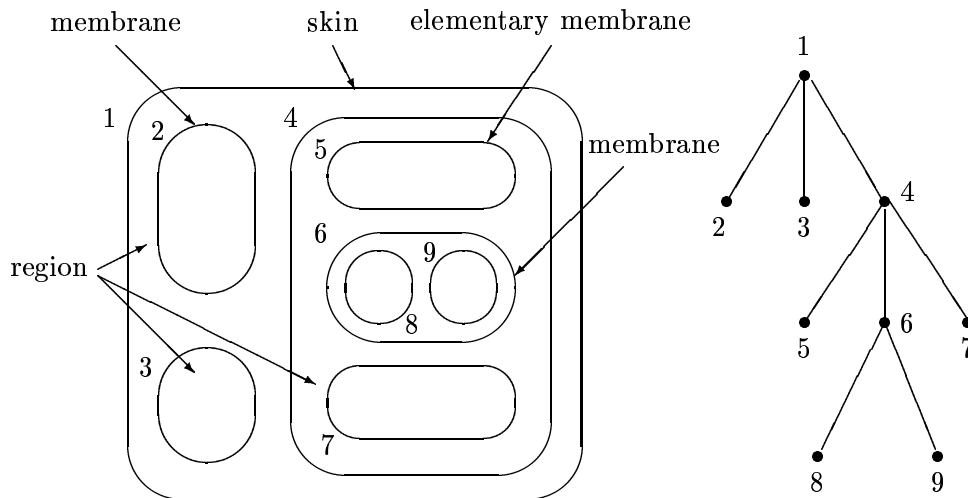
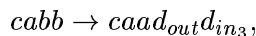


Figure 4: A membrane structure and its associated tree

In the regions of a membrane structure we place *multisets of objects*. A multiset is a usual set with multiplicities associated with each object, in the form of natural numbers; the meaning is that each object can appear in a number of identical copies in a given region. In turn, for the beginning, the objects are supposed to be *atomic*, in the etimological sense, without “parts”, hence we can identify them by symbols from a given alphabet (we always work with finitely many types of objects, that is, with multisets over a finite support-set).

The objects evolve by means of given *rules*, which are associated with the regions (the intuition is that each region has specific chemical reaction conditions or specific enzymes, hence the rules from a region cannot necessarily act also elsewhere). These rules specify both object transformation and object transfer from a region to another one. The passing of an object through a membrane is called *communication*.

Here is a typical rule



with the following meaning: one copy of the *catalyst* c (note that it is reproduced after the “reaction”) together with one copy of object a and two copies of object b react together and produce one copy of c , two of a , and two copies of the object d ; one of these latter objects is sent out of the region where the rule is applied, while the second copy is send to the adjacently inner membrane with the label 3, if such a membrane exists; the objects c, a, a remain in the same membrane (it is supposed that they have associated the communication command *here*, but we do not explicitly write this indication); if there is no membrane with label 3 directly inside the membrane where the rule is to be applied, then the rule cannot be applied. By a command *out*, an object can be also sent outside the skin membrane, hence it leaves the system and never comes back.

Therefore, the rules are multiset processing; in the previous case, the multiset represented by $cabb$ is subtracted from the multiset of objects in a given region, objects caa are added to

the multiset in that region, while copies of d are added to the multisets in the upper and lower regions.

To a rule as above we can also add the *action-symbol* δ ; for example, in the previous case, we write $cabb \rightarrow caad_{out}d_{in_3}\delta$. When a rule which contains δ is used, then the membrane is *dissolved*, all its objects remain free in the immediately upper membrane and its rules are lost. The skin membrane is never dissolved. A sort of dual action can also be considered, τ , which makes a membrane thicker, thus inhibiting the communication through it; a thicker membrane can be open again to communications after using a rule which contains the symbol δ . By using the actions δ, τ one can control the communication through membranes in a very effective way – see Section 7 below.

The rules are used *in a nondeterministic maximally parallel manner*: the objects to evolve and the rules to be applied to them are chosen in a non-deterministic manner, but all objects which can evolve at a given step should do it. Sometimes, a priority relation among rules is considered, hence the rules to be used and the objects to be processed are selected in such a way that only rules which have a maximal priority among the applicable rules are used.

The membrane structure together with the multisets of objects and the sets of evolution rules present in its regions constitute a *P system*. The membrane structure and the objects define a *configuration* of a given P system. By using the rules in the way suggested above, we can define *transitions* among configurations. A sequence of transitions is called a *computation*. We accept as successful computations only the *halting* ones, those which reach a configuration where no further rule can be applied.

With a successful computation we can associate a *result*, for instance, by counting the objects present in the halting configuration in a specified elementary membrane, i_o . More specifically, we can use a P system for solving three types of tasks: as a *generative* device (start from an initial configuration and collect all vectors of natural numbers describing the multiplicities of objects present in membrane i_o at the end of all successful computations), as a *computing* device (start with some input placed in an initial configuration and read the output at the end of a successful computation, by considering the contents of membrane i_o), and as a *decidability* device (introduce a problem in an initial configuration, in a specific encoding, and look for the answer after a specified number of steps, in the contents of membrane i_o).

Of course, instead of using an elementary membrane for reading the output, we can consider the objects which leave the skin membrane and we can “compose” the result by using them.

Some of these notions will be partially given in a more formal manner in the subsequent sections. We anticipate an important observation: *various computing devices as sketched above are computationally complete, in the Turing sense; for instance, in the generative case, they can generate all recursively enumerable sets of vectors of natural numbers.*

This means computing *competence*. What about computing *performances*, in the sense of the (time) complexity of solving problems? In Section 10 we will see that certain classes of P systems, with an enhanced parallelism, are able to solve NP-complete problems in linear time. Of course, this means trading space for time and using an exponential space, in the form of membranes and objects present in the configurations of the system during a computation (this is usual in Molecular Computing, for instance, in DNA Computing, where we can count on the massive parallelism made possible by the fact that billions of molecules can find room in a tinny test tube). The usefulness of P systems from a practical computational point of view is an important topic, so we devote to it the next section.

4 Membrane Computing and Natural Computing

The details from the biochemistry of (animal and vegetal, alike) membranes sketched in Section 2 only show that the ingredients we use in P systems, mainly in what concerns the way of communicating through membranes, are “realistic”, in the sense that they have a resemblance with actual biochemical facts. However, this means nothing in what concerns the possible implementations of the model.

Up to now, there was no attempt to carry out a computation, even for solving a toy problem, by using actual bio-membranes, handled *in vitro* or just observed, maybe also influenced, *in vivo*.

In fact, there is a fundamental question here, about the right way of looking for implementations of P systems. The situation corresponds to a basic dilemma in Natural Computing, and it is illustrated⁴ in Figure 5.

Computing with membranes starts from the analogy of processes which take place in the complex structure of a living cell with a computing process. In the style of other branches of Natural Computing, we learn from (alive) nature new computing models and strategies (let us say, paradigms), but it is not clear in this moment whether or not we have to go back to biochemistry for implementing the new computing models (like in DNA Computing) or to the electronic computer (to the general purpose one, or to a specially designed one) for implementation (like in Neural Networks and Genetic Algorithms). The papers [P4: Baranda et al., 2000], [P7: Ciobanu, Paraschiv, 2000], [P24: Malița, 2000], [P48: Suzuki, Tanaka, 2000], and [P50: Ștefan, 2000] have attempted to simulate P systems on the electronic computer, but the results are not yet answering the question of the practical usefulness of this approach.

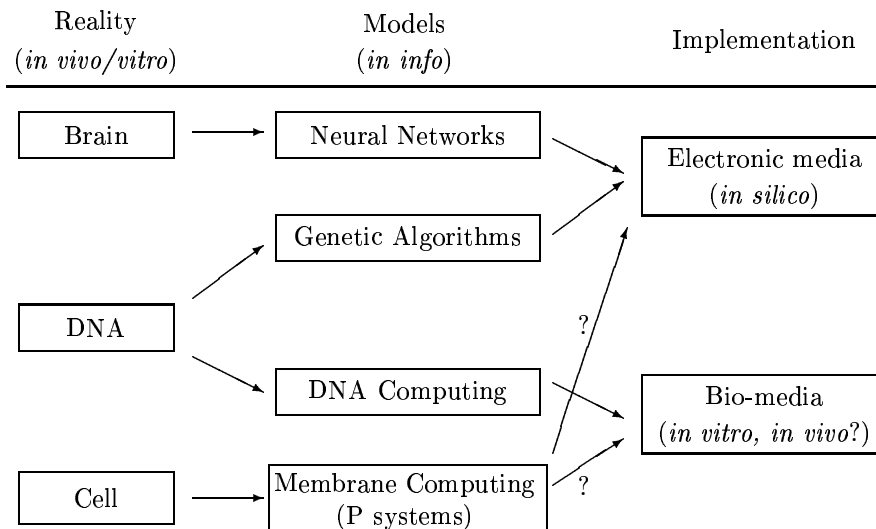


Figure 5: The four bio-domains of Natural Computing

Of course, we have also to be prepared for the case that no implementation of P systems will be done (that is, no implementation of a real practical interest), and that the attempt will remain forever *in info*, as a subject for mathematical investigations. Anyway, it is too early to ask such questions, serious interdisciplinary investigations should precede any sensible answer.

⁴For our discussion here, we interpret the sintagm “Natural Computing” in the sense of “biologically inspired computing”, without including the Quantum Computing, as done, for instance, by the Natural Computing column from *Bulletin of the EATCS*.

5 A Formal Definition of a P System

In this section we go closer to mathematics and introduce a formal counterpart of the computing cell as sketched in Section 3.

A membrane structure can be mathematically represented by a tree, in the natural way, or by a string of matching parentheses. The tree of the structure in Figure 1 is given in the same figure, while the parenthetic representation of that structure is the following:

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

The tree representation makes possible considering various parameters, such as the *depth* of the membrane structure, and also suggests considering membrane structures of particular types (described by linear trees, star trees, etc).

A multiset over an alphabet $V = \{a_1, \dots, a_n\}$ is a mapping M from V to \mathbf{N} , the set of natural numbers ($M(a_i)$ indicates the number of copies – the multiplicity – of the element a_i in the multiset M), and it can be represented by any string $w \in V^*$ such that⁵ $\Psi_V(w) = (M(a_1), \dots, M(a_n))$. Operations with multisets are defined in the natural manner.

With these simple prerequisites, we can define a P system (of *degree* m , $m \geq 1$) as a construct

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

where:

- (i) V is an alphabet; its elements are called *objects*;
- (ii) $T \subseteq V$ (the *output* alphabet);
- (iii) $C \subseteq V, C \cap T = \emptyset$ (*catalysts*);
- (iv) μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) labeled in a one-to-one manner with elements of a given set Λ ; in this section we use the labels $1, 2, \dots, m$;
- (v) $w_i, 1 \leq i \leq m$, are strings which represent multisets over V associated with the regions $1, 2, \dots, m$ of μ ;
- (vi) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over V associated with the regions $1, 2, \dots, m$ of μ ; ρ_i is a partial order relation over $R_i, 1 \leq i \leq m$, specifying a *priority* relation among rules of R_i .

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 $u \rightarrow v$.

- (vii) i_o is a number between 1 and m (the *output* membrane).

When presenting the evolution rules, the indication “here” is in general 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 $ca \rightarrow cv$ or $ca \rightarrow cv\delta$,

⁵The elementary notions of formal language theory which we use here can be found, e.g., in Rozenberg, Salomaa, 1997. In particular, Ψ_V is the *Parikh mapping* associated with V : for $a \in V$ and $x \in V^*$ (where V^* denoted the set of all strings over V , including the empty one, λ), we denote by $|x|_a$ the number of occurrences of a in x ; then $\Psi_V(x) = (|x|_{a_1}, \dots, |x|_{a_n})$.

where $c \in C$, $a \in V - C$, and v contains no catalyst; moreover, no other evolution rules contain catalysts (there is no rule of the form $c \rightarrow v$ or $a \rightarrow v_1cv_2$, for $c \in C$).

The $(m + 1)$ -tuple (μ, w_1, \dots, w_m) constitutes the *initial configuration* of Π . In general, any sequence $(\mu', w'_1, \dots, w'_k)$, with μ' a membrane structure obtained by removing from μ all membranes different from i_1, \dots, i_k (of course, the skin membrane is not removed), with w'_j strings over V , $1 \leq j \leq k$, and $\{i_1, \dots, i_k\} \subseteq \{1, 2, \dots, m\}$, is called a *configuration* of Π .

It should be noted the important detail that the membranes preserve the initial labeling in all subsequent configurations; in this way, the correspondence between membranes, multisets of objects, and sets of evolution rules is well specified by the subscripts of these elements.

For two configurations $C_1 = (\mu', w'_{i_1}, \dots, w'_{i_k})$, $C_2 = (\mu'', w''_{j_1}, \dots, w''_{j_l})$ 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}, \dots, R_{i_k} in the following manner (rather than a completely cumbersome formal definition we prefer an informal one, explained by examples).

Consider a rule $u \rightarrow v$ in a set R_{i_t} . We look to the region of μ' associated with the membrane i_t . If the objects mentioned by u , with the multiplicities at least as large as specified by u , appear in w'_{i_t} , then these objects can evolve according to the rule $u \rightarrow v$. The rule can be used only if no rule of a higher priority exists in R_{i_t} and can be applied at the same time with $u \rightarrow v$. More precisely, we start to examine the rules in the decreasing order of their priority and assign objects to them. A rule can be used only when there are copies of the objects whose evolution it describes and which were not “consumed” by rules of a higher priority and, moreover, there is no rule of a higher priority, irrespective which objects it involves, which is applicable at the same step. Therefore, all objects to which a rule *can* be applied *must* be the subject of a rule application. All objects in u are “consumed” by using the rule $u \rightarrow v$.

The result of using the rule is determined by v . If an object appears in v in the form a_{here} , then it will remain in the same region i_t . If an object appears in v in the form a_{out} , then a will exit the membrane i_t and will become an element of the region which surrounds membrane i_t . In this way, it is possible that an object leaves the system: if it goes outside the skin of the system, then it never comes back. If an object appears in the form a_{in_q} , then a will be added to the multiset from membrane q , providing that a is adjacent to the membrane q . If a_{in_q} appears in v and membrane q is not one of the membranes delimiting “from below” the region i_t , then the application of the rule is not allowed.

If the symbol δ appears in v , then membrane i_t is removed (we say *dissolved*) and at the same time the set of rules R_{i_t} (and its associated priority relation) is removed. The multiset from membrane i_t is added (in the sense of multisets union) to the multiset associated with the region which was directly external to membrane i_t . We do not allow the dissolution of the skin membrane, because this means that the whole “cell” is lost, we do no longer have a correct configuration of the system.

All these operations are performed in parallel, for all possible applicable rules $u \rightarrow v$, for all occurrences of multisets u in the regions associated with the rules, for all regions at the same time. No contradiction appears because of multiple membrane dissolution, or because of simultaneous appearance of symbols of the form a_{out} and δ . If at the same step we have a_{in_i} outside a membrane i and δ inside this membrane, then, because of the simultaneity of performing these operations, again no contradiction appears: we assume that a is introduced in membrane i at the same time when it is dissolved, thus a will remain in the region surrounding membrane i ; that is, from the point of view of a , the effect of a_{in_i} in the region outside membrane i and δ in membrane i is a_{here} .

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, and if the membrane i_o is present as an elementary one in the last configuration of the computation. (Note that the output membrane was not necessarily an elementary one in the initial configuration.) The result of a successful computation is $\Psi_T(w)$, where w describes the multiset of objects from T present in the output membrane in a halting configuration. The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$ (from “Parikh set”) and we say that it is *generated* by Π .

The family of sets of vectors of natural numbers $Ps(\Pi)$ generated by P systems with priority, catalysts, and the membrane dissolving action, and of degree at most $m, m \geq 1$, using target indications of the form *here, out, in_j* (but not using the action τ , which will be defined in Section 7), is denoted by $NP_m(Pri, Cat, tar, \delta, n\tau)$; when one of the features $\alpha \in \{Pri, Cat, \delta\}$ is not present, we replace it with $n\alpha$. When the number of membranes is not bounded, we replace the subscript m with $*$.

6 Examples

Before recalling some results about the computing power of P systems of the types introduced in the previous section or involving further ingredients, we shall consider three examples for the basic variant considered above, one for each type of tasks a P system can solve.

Example 1. (Generative task) Consider the P system of degree 4

$$\begin{aligned} \Pi_1 &= (V, T, C, \mu, w_1, w_2, w_3, w_4, (R_1, \rho_1), (R_2, \rho_2), (R_3, \rho_3), (R_4, \rho_4), 4), \\ V &= \{a, b, b', c, e, f\}, T = \{e\}, C = \emptyset, \\ \mu &= [{}_1[{}_2[{}_3[{}_4]_2]_1], \\ w_1 &= \lambda, R_1 = \emptyset, \rho_1 = \emptyset, \\ w_2 &= \lambda, R_2 = \{b' \rightarrow b, b \rightarrow be_{in_4}, r_1 : ff \rightarrow f, r_2 : f \rightarrow \delta\}, \rho_2 = \{r_1 > r_2\}, \\ w_3 &= af, R_3 = \{a \rightarrow ab', a \rightarrow b'\delta, f \rightarrow ff\}, \rho_3 = \emptyset, \\ w_4 &= \lambda, R_4 = \emptyset, \rho_4 = \emptyset. \end{aligned}$$

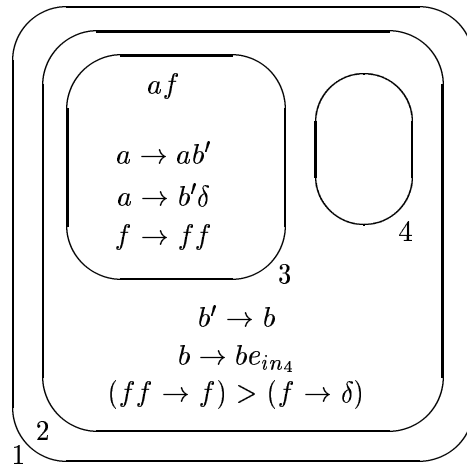


Figure 6: A P system generating $n^2, n \geq 1$

The initial configuration of the system is given in Figure 6. No object is present in membrane 2, hence no rule can be applied here. The only possibility is to start in membrane 3, using the

objects a, f , each of them present in one copy. Using the rules $a \rightarrow ab', f \rightarrow ff$, in parallel for all occurrences of a and f currently available, after n steps, $n \geq 0$, we get n occurrences of b' and 2^n occurrences of f . In any moment, instead of $a \rightarrow ab'$ we can use $a \rightarrow b'\delta$ (note that we always have only one copy of a). In that moment we have $n + 1$ occurrences of b' and 2^{n+1} occurrences of f and we dissolve membrane 3.

The rules of the former membrane 3 are lost, the rules of membrane 2 can now be applied to the objects left free in its region. Due to the priority relation, we have to use the rule $ff \rightarrow f$ as much as possible. In one step, we pass from b'^{n+1} to b^{n+1} , while the number of f occurrences is divided by two. In the next step, $n + 1$ occurrences of e are introduced in membrane 4: each occurrence of the symbol b introduces one occurrence of e . At the same time, the number of f occurrences is divided again by two. We continue. At each step, further $n + 1$ occurrences of e are introduced in the output membrane. This can be done $n + 1$ steps: n times when the rule $ff \rightarrow f$ is used (thus diminishing the number of f occurrences to one), and one when using the rule $f \rightarrow \delta$ (it may now be used). In this moment, membrane 2 is dissolved, which entails the fact that its rules are removed. No further step is possible, the computation stops.

In total, we have $(n + 1)(n + 1)$ copies of e in membrane 4, for some $n \geq 0$, that is,

$$Ps(\Pi_1) = \{(n^2) \mid n \geq 1\}.$$

Example 2. (Computing task) The previous system can easily be modified in such a way that a number n is introduced in the initial configuration, as the number of copies of the object a present in membrane 3, and the system stops after computing the square of n . We present such a system directly in a pictorial representation, in Figure 7 (where c is a catalyst). The reader can easily convince itself that the result of a halting computation is n^2 (note that the catalyst is present in only one copy, hence changing a into b takes n steps, while at the last step one introduces two copies of b ; from that step on, this system works exactly as Π_1 , with the mentioning that if membrane 3 is prematurely dissolved, then the computation never ends, because of the rule $a \rightarrow a$ present in membrane 2).

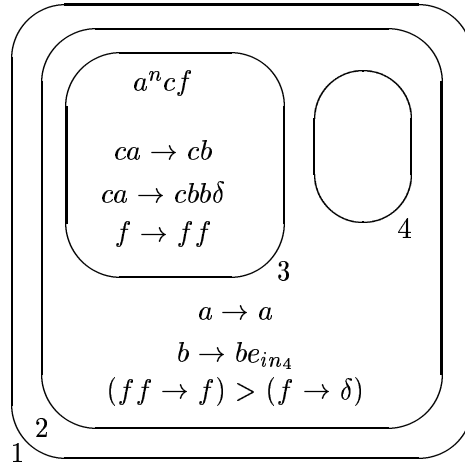


Figure 7: A P system computing $n \rightarrow n^2$

Example 3. (Decidability task) We introduce in the input configuration two numbers, n and k , and ask whether or not n is a multiple of k . The answer is received outside the system

(hence the output membrane is not specified). The system is the following (of degree 2):

$$\begin{aligned}\Pi_3 &= (\{a, b, b', d, err, no, yes\}, \{err, no, yes\}, \emptyset, [_1[_2]_2]_1, \lambda, a^n b^k d, (R_1, \rho_1), (R_2, \emptyset)), \\ R_1 &= \{r_1 : a \rightarrow a \text{ err}_{out}, r_2 : dbb' \rightarrow no_{out}, r_3 : d \rightarrow yes_{out}\}, \\ \rho_1 &= \{r_1 > r_2, r_2 > r_3\}, \\ R_2 &= \{ab \rightarrow b', ab' \rightarrow b, d \rightarrow d, d \rightarrow d\delta\}.\end{aligned}$$

In membrane 2 we subtract k from n , repeatedly (by the rules $ab \rightarrow b'$, $ab' \rightarrow b$: at each step, k copies of a are removed, while b is reproduced, primed or not primed, alternating the priming from a step to another one). At any time, the symbol d will introduce δ and this membrane is dissolved.

If an occurrence of a arrives in membrane 1, then the computation never ends. Therefore, we have to dissolve membrane 2 only after exhausting the n occurrences of a . If n is not a multiple of k , then we have both occurrences of b and of b' present in membrane 1 and the rule $dbb' \rightarrow no_{out}$ can be used, otherwise this rule cannot be used and the lower priority rule r_3 is used, sending outside the object yes .

Note that these rules can be used at most once, because we have only one occurrence of d , hence the computation stops after sending out of the system any of the “messages” err , no , yes .

7 Computational Completeness

At the end of Section 3 we have mentioned that P systems as introduced above are equivalent with Turing Machines. More precisely, we have the following theorem (the family of recursively enumerable sets of vectors of natural numbers is denoted⁶ by $PsRE$):

Theorem 1. $NP_2(Pri, Cat, tar, n\delta, n\tau) = PsRE$.

The proof of this result is rather complex (see [P32: Gh. Păun, 1998]). As many proofs of computational completeness of various classes of P systems, it is based on the following ideas: one knows that the recursively enumerable languages are characterized by *matrix grammars with appearance checking*⁷; therefore, $PsRE$ is equal to the family of Parikh sets of languages generated by matrix grammars with appearance checking; for these grammars, there is a *binary normal form* (see Lemma 1.3.7 in Dassow, Gh. Păun, 1989), where each matrix has only two rules, with separate sets of nonterminals in rules in the first position and rules in the second position of matrices; moreover, there always exists only one nonterminal which can be rewritten by the first rule of matrices; starting from such a grammar, one constructs a P system which computes (at the end of halting computations) exactly the Parikh vectors of terminal strings generated by the grammar.

⁶We denote by RE the family of recursively enumerable languages, that is, the languages generated by type-0 Chomsky grammars and recognized by Turing machines; then, $PsRE$ is exactly the set of Parikh images $\Psi_V(L)$, of languages $L \in RE$, $L \subseteq V^*$.

⁷Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N$, $x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices). For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ in M and the strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i , and the rule $A_i \rightarrow x_i$ appears in F . (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac} . It is known that $MAT_{ac} = RE$.

The result in Theorem 1 is obtained at the price of using two very powerful features, priority relations among evolution rules and target commands of the form in_j , with the precise indication of the target membrane, j . While the priority can be claimed to correspond to different reactivities of different chemical compounds, the indication of the target membrane only remotely corresponds to the selective protein channels. On the good side, we do not make use of the actions δ, τ .

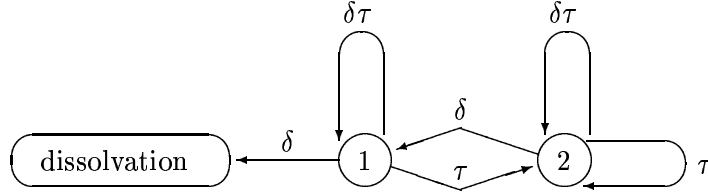


Figure 8: The effect of actions δ, τ

In the aim of obtaining more “realistic” systems, we can consider weaker communication commands. The weakest one is to add no label to in : if an object a_{in} is introduced in some region of a system, then a will go to any of the adjacent lower membranes, nondeterministically chosen; if no inner membrane exists, then a rule which introduces a_{in} cannot be used (this might be considered as related to the passing of chemical compounds through membranes because of concentration difference, without any selectivity of different membranes).

An intermediate possibility is to associate both with objects and membranes *electrical charges*, indicated by $+, -, 0$ (positive, negative, neutral). The charges of membranes are given in the initial configuration and are not changed during computations, the charge of objects are given by the evolution rules, in the form $a \rightarrow b^+d^-$. A charged object will immediately go into one of the directly lower membrane of the opposite polarization, the neutral objects remain in the same region or will exit it, depending on having associated one of the indications *here* and *out*. After moving a charged object through a membrane, its polarization is set to neutral.

Because communication by means of in commands and by electrical charges is expectedly weaker than communication by commands of the form in_j , we compensate the loss in power by making use of the actions δ, τ in order to control the membrane thickness (hence permeability). This is done as follows. Initially, all membranes are considered of thickness 1. If a rule in a membrane of thickness 1 introduces the symbol τ , then the membrane becomes of thickness 2. A membrane of thickness 2 does not become thicker by using further rules which introduce the symbol τ , but no object can enter or exit it. If a rule which introduces the symbol δ is used in a membrane of thickness 1, then the membrane is dissolved; if the membrane had thickness 2, then it returns to thickness 1. If at the same step one uses rules which introduce both δ and τ in the same membrane, then the membrane does not change its thickness. These actions of the symbols δ, τ are illustrated in Figure 8.

The use of commands *here, out, in* is indicated in the notations of families of generated sets of vectors by writing i/o , and the use of electrical charges is indicated by writing \pm instead of tar , respectively.

Systems with polarized membranes were introduced in [P34: Gh. Păun, 2000a], where it is proved that $NP_*(nPri, Cat, \pm, \delta, \tau) = PsRE$. The result was recently considerably improved in [P14: Freund, Martin-Vide, Gh. Păun, 2000]:

Theorem 2. $NP_5(nPri, Cat, i/o, \delta, \tau) = PsRE$.

The proof uses an idea which can keep the number of membranes bounded by a reasonably

small threshold: when simulating a matrix grammar by a P system it is not necessary to deal separately with all matrices, but only with nonterminals which appear in rules which are used in the appearance checking mode. Because the number of such nonterminals can be bounded (by three – see Freund, 2000, which has improved the bound six from Gh. Păun, 1984), the number of membranes is drastically bounded.

Instead of using catalysts as considered up to now we can also use a sort of “catalysts with a short term memory”. Such catalysts (we call them *bi-stable*) have two states each, c and \bar{c} , and they can enter rules of the forms $ca \rightarrow \bar{c}v, \bar{c}a \rightarrow cv$ (always changing from c to \bar{c} and back). We write $2Cat$ instead of Cat when denoting the generated families of vector sets. The proof of the next result can be found in [P44: Gh. Păun, Yu, 1999]; note that this time the hierarchy on the number of membranes collapses at the first level, while the actions δ, τ are not used, which shows the excessive power of bi-stable catalysts.

Theorem 3. $NP_1(nPri, 2Cat, i/o, n\delta, n\tau) = PsRE$.

8 P Systems with String-Objects

In P systems with symbol-objects, as discussed in the previous sections, we use a rather non-economical representation of numbers, the base one. Having in mind that many of the objects present in cells are complex molecules, even strings, such as the DNA molecules, it is natural to consider also P systems with the objects represented by strings. Then, the evolution rules should be string processing operations.

An important distinction here is whether or not we take into account the multiplicities of strings. Because besides the information contained in their Parikh vector, the strings also contain the intrinsic syntactic information (and because dealing with multisets at the cellular level, although biochemically motivated, is practically difficult), it is maybe better not to use multisets, but usual sets of strings (languages). Anyway, this also depends on the used operations, on whether or not they are able to increase/decrease the number of copies of strings in the regions of the system.

The simplest case is that of *rewriting P systems*, which use rules of the form $(X \rightarrow v; tar)$, where $X \rightarrow v$ is a usual context-free rule and tar is a target indication, one of *here, out, in_j*, specifying the region where the result of rewriting should go. Because the rewriting does not modify the number of strings, we have to deal with languages, not with multisets of strings.

The structure and the functioning of a rewriting P system are defined in the natural way. We denote by $RLP_m(Pri, i/o)$ the family of languages generated by rewriting P systems with at most m membranes, using priorities among the rewriting rules and target indications *here, out, in* (and no other ingredients, such as the dissolving action, etc). The proof of the following result can be found in [P19: Krishna, Rama, 2000a], where one improves a similar result, about systems with three membranes, from [P32: Gh. Păun, 1998]:

Theorem 4. $RLP_2(Pri, i/o) = RE$.

A more attractive variant seems to be the *splicing P systems*, where the objects are processed by the splicing operation, introduced in (Head, 1987) as a formal model of the DNA recombination under the influence of restriction enzymes and ligases (see a comprehensive investigation of splicing systems in (Gh. Păun, Rozenberg, Salomaa, 1998)).

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

$$(x, y) \vdash_r (w, z) \quad \text{iff} \quad x = x_1 u_1 u_2 x_2, \quad y = y_1 u_3 u_4 y_2, \quad w = x_1 u_1 u_4 y_2, \quad z = y_1 u_3 u_2 x_2,$$

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 one recombines the fragments obtained in this way.)

A *splicing P system* (of degree $m, m \geq 1$) is a construct

$$\Pi = (V, T, \mu, L_1, \dots, L_m, R_1, \dots, R_m),$$

where:

- (i) V is an alphabet;
- (ii) $T \subseteq V$ (the *output* alphabet);
- (iv) μ is a membrane structure consisting of m membranes (labeled with $1, 2, \dots, m$);
- (v) $L_i, 1 \leq i \leq m$, are languages over V associated with the regions $1, 2, \dots, m$ of μ ;
- (vi) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* associated with the regions $1, 2, \dots, m$ of μ , given in the following form: $(r; tar_1, tar_2)$, where $r = u_1 \# u_2 \$ u_3 \# u_4$ is a usual splicing rule over V and $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

Note that, as usual in splicing systems, when a string is present in a region of our system, it is assumed to appear in arbitrarily many copies.

The transition among configurations of Π is defined by applying the splicing rules from each region of μ , in parallel, to all possible strings from the corresponding regions, and following the target indications associated with the rules. More specifically, if x, y are from region i and $(r = u_1 \# u_2 \$ u_3 \# u_4; tar_1, tar_2) \in R_i$ 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. If $tar_j = here$, then the string remains in region i , if $tar_j = out$, then the string is moved to the region surrounding membrane i (maybe, in this way the string leaves the system), if $tar_j = in_k$, then the string is moved to the region k , providing that this is immediately below; if not, then the rule cannot be applied. Note that the strings x, y are still available in region i , because we have supposed that they appear in arbitrarily many copies (an arbitrarily large number of them were spliced, arbitrarily many remain), but if a string w, z , obtained by splicing, is sent out of region i , then no copy of it remains here.

The result of a computation consists of all strings over T which are sent out of the system at any time during the computation. We denote by $L(\Pi)$ the language of all strings of this type (we say that $L(\Pi)$ is *generated* by Π). Note that in the case of splicing P systems we do not consider halting computations, but we leave the process to continue forever and we just observe the system from outside and collect the terminal strings which leave it.

We denote by $SLP_m(tar)$ the family of languages $L(\Pi)$ generated by splicing P systems as above, of degree at most $m, m \geq 1$; tar is replaced by i/o when appropriately.

Splicing P systems as above, with three membranes either arranged in two levels or in three levels, were shown in [P42: Gh. Păun, Yokomori, 1999] to characterize the recursively enumerable languages. The result was improved in [P31: M. Păun, A. Păun, 2000] – two membranes suffice:

Theorem 5. $SLP_2(i/o) = RE$.

9 P Systems with Worm-Objects

In P systems with symbol-objects we work with multisets and the result of a computation is a natural number or a vector of natural numbers; in the case of P systems with string-objects we work with sets of strings and the result of a computation is a string. We can combine the two ideas: we can work with multisets of strings and consider as the result of a computation the number of strings present in the halting configuration in a given membrane or sent out of the system during the computation. To this aim, we need operations with strings which can increase and decrease the number of occurrences of strings.

The following four operations were considered in [P6: Castellanos, Gh. Păun, Rodriguez-Paton, 2000] (they are slight variants of the operations used in (Sipper, 1995)):

1. *Replication*. If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \rightarrow u_1|u_2$ is called a *replication rule*⁸. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Rightarrow_r (w_2, w_3)$ (and we say that w_1 is replicated with respect to rule r) if $w_1 = x_1ax_2$, $w_2 = x_1u_1x_2$, $w_3 = x_1u_2x_2$, for some $x_1, x_2 \in V^*$.
2. *Splitting*. If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \rightarrow u_1|u_2$ is called a *splitting rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Rightarrow_r (w_2, w_3)$ (and we say that w_1 is splitted with respect to rule r) if $w_1 = x_1ax_2$, $w_2 = x_1u_1$, $w_3 = u_2x_2$, for some $x_1, x_2 \in V^*$.
3. *Mutation*. A *mutation rule* is a context-free rewriting rule, $r : a \rightarrow u$, over V . For strings $w_1, w_2 \in V^+$ we write $w_1 \Rightarrow_r w_2$ if $w_1 = x_1ax_2$, $w_2 = x_1ux_2$, for some $x_1, x_2 \in V^*$.
4. *Recombination*. Consider a string $z \in V^+$ (as a *crossing-over block*) and four strings $w_1, w_2, w_3, w_4 \in V^+$. We write $(w_1, w_2) \Rightarrow_z (w_3, w_4)$ if $w_1 = x_1zx_2$, $w_2 = y_1zy_2$, and $w_3 = x_1zy_2$, $w_4 = y_1zx_2$, for some $x_1, x_2, y_1, y_2 \in V^*$.

Let us note that replication and splitting increase the number of strings, while mutation and recombination not; we can also decrease the number of objects, by sending strings out of the system.

A *P system* (of degree $m, m \geq 1$) with worm-objects is a construct

$$\Pi = (V, \mu, A_1, \dots, A_m, (R_1, S_1, M_1, C_1), \dots, (R_m, S_m, M_m, C_m), i_o),$$

where:

- V is an alphabet;
- μ is a membrane structure of degree m , with the membranes labeled with $1, 2, \dots, m$;
- A_1, \dots, A_m are multisets of finite support over V^* , associated with the regions of μ ;
- for each $1 \leq i \leq m$, R_i, S_i, M_i, C_i are finite sets of replication rules, splitting rules, mutation rules, and crossing-over blocks, respectively, given in the following forms:
 - a. replication rules: $(a \rightarrow u_1|u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
 - b. splitting rules: $(a \rightarrow u_1|u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
 - c. mutation rules: $(a \rightarrow u; tar)$, for $tar \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
 - d. crossing-over blocks: $(z; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
- $i_o \in \{1, 2, \dots, m\}$ specifies the *output membrane* of the system; it should be an elementary membrane of μ .

⁸ V^+ denotes the set of non-empty strings over V , that is, $V^+ = V^* - \{\lambda\}$.

The $(n + 1)$ -tuple (μ, A_1, \dots, A_m) constitutes the *initial configuration* of the system. By applying the operations defined by the components $(R_i, S_i, M_i, C_i), 1 \leq i \leq m$, we can define transitions from a configuration to another one. This is done as usual in P systems area, according to the following additional rules: 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 accordingly increased. A string is processed by only one operation. For instance, we cannot apply two mutation rules, or a mutation rule and a replication one, to the same string.

The result of a halting computation consists of the number of strings in region i_o at the end of the computation. A non-halting computation provides no output. For a system Π , we denote by $N(\Pi)$ the set of numbers computed in this way. By $NWP_m, m \geq 1$, we denote the sets of numbers computed by all P systems with worm-objects with at most m membranes. The family of all recursively enumerable sets of natural numbers is denoted by nRE .

In [P6: Castellanos, Gh. Păun, Rodriguez-Paton, 2000] one characterizes nRE without obtaining a bound on the number of membranes, but the following result was proved in [P27: Martin-Vide, Gh. Păun, 2000a]:

Theorem 6. $nRE = NWP_6$.

Note that in this moment this is “the highest” hierarchy known in this area for families which characterize RE, nRE , or $PsRE$. It is an *open problem* whether or not the bound 6 is optimal; we expect a negative answer.

10 Solving NP in P by P

Let us now pass to a fundamental question: are P systems computationally useful? From a theoretical point of view, the answer is affirmative: in the framework of P systems we can naturally create an exponential working space and this can be used for solving hard problems in a feasible time. Exponentially many objects can be produced in all types of P systems – see the very first example from Section 6. However, such a work space is not sufficient: an important theorem from [P54: Zandron, Ferretti, Mauri, 2000d] says that a deterministic P system with symbol objects (and without membrane division as we will consider below) can be simulated by a deterministic Turing machine with a polynomial slowdown. Therefore, a more powerful way of producing exponential space is necessary. We will illustrate this possibility with the case of P systems with worm-objects.

This was already shown in [P6: Castellanos, Gh. Păun, Rodriguez-Paton, 2000] for the Hamiltonian Path Problem (HPP), the very problem used by L. Adleman in his history-making experiment, (Adleman, 1994). We recall some details from [P6: Castellanos, Gh. Păun, Rodriguez-Paton, 2000].

Consider a directed graph $\gamma = (U, E)$ without cycles $(i, i), i \in U$, and two distinct nodes from U , i_{in} and i_{out} . Assume that U contains n nodes, identified with the numbers $1, 2, \dots, n$ (hence the Hamiltonian paths will consist of $n - 1$ arcs) and that the maximum outdegree of the graph (the number of arcs having the origin in a given node) is equal to k . By a simple renumbering, assume that $i_{in} = 1$ and that $i_{out} = n$.

We construct the P system Π_γ (of degree n), associated with γ , with the following components:

$$\begin{aligned} V &= \{ \langle i, r \rangle, [i, r], \langle i, r; j_1, \dots, j_s \rangle \mid 1 \leq i \leq n, 0 \leq r \leq n, 1 \leq s \leq k, \\ &\quad \{j_1, \dots, j_s\} \subseteq \{j \in U \mid (i, j) \in E\} \}, \\ \mu &= [_{n-1} [_{n-1}]_n]_n [_{n-2} \dots [_{2} [_{0}]_2 \dots]_{n-2}]_{n-1}, \end{aligned}$$

$$\begin{aligned}
A_0 &= \{(\langle 1, 0 \rangle, 1)\}; \text{ all other initial multisets are empty,} \\
R_0 &= \{(\langle i, r \rangle \rightarrow [i, r] \langle j_1, r + 1 \rangle \mid \langle i, r; j_2, \dots, j_s \rangle; \textit{here, here}), \\
&\quad (\langle i, r; j_h, \dots, j_s \rangle \rightarrow [i, r] \langle j_h, r + 1 \rangle \mid \langle i, r; j_{h+1}, \dots, j_s \rangle; \textit{here, here}), \\
&\quad (\langle i, r; j_{s-1}, j_s \rangle \rightarrow [i, r] \langle j_{s-1}, r + 1 \rangle \mid [i, r] \langle j_s, r + 1 \rangle; \textit{here, here}) \mid \\
&\quad \text{for all } 1 \leq i \leq n - 1, 0 \leq r \leq n - 2, \text{ where} \\
&\quad j_1, \dots, j_s \text{ are all the nodes such that } (i, j_l) \in E, 1 \leq l \leq s\}, \\
M_0 &= \{(\langle n, n - 1 \rangle \rightarrow [n, n - 1]; \textit{out})\}, S_0 = C_0 = \emptyset, \\
R_i &= S_i = C_i = \emptyset, \text{ for all } 2 \leq i \leq n - 1, \\
M_i &= \{([i, j] \rightarrow [i, j]; \textit{out}) \mid 1 \leq j \leq n - 1\}, \text{ for all } 2 \leq i \leq n - 2, \\
M_{n-1} &= \{([n - 1, j] \rightarrow [n - 1, j]; \textit{in}_n) \mid 1 \leq j \leq n - 2\}, \\
R_n &= S_n = M_n = C_n = \emptyset.
\end{aligned}$$

(Note that only replication and mutation rules are used.)

The membrane structure of this system is given in Figure 9; membrane n is the output one. We start from the unique object $\langle 1, 0 \rangle$, present in the inner membrane, that with label 0, by repeatedly using replication rules. These rules always prolong a string which represents a path in the graph starting from node 1. No path can be continued if either it reaches node n or we have already made $n - 1$ steps (hence the path already passes through n nodes and any further step will surely repeat a node). In this way, we can generate all paths in the graph γ starting in node 1 and of length (as the number of arcs) at most $n - 1$.

Only strings which end with $\langle n, n - 1 \rangle$ can be sent outside membrane 0 (this means that the corresponding paths end in node n).

In each membrane i , $2 \leq i \leq n - 2$, a string can be only processed if it contains a symbol of the form $[i, t]$, for some $1 \leq t \leq n - 1$. (This means that node i was visited at time t .) If this is the case, then the string is sent unmodified to the next membrane. If a string reaches the skin membrane, then it can be sent to the output membrane if and only if it contains a symbol $[n - 1, j]$ (this means that the corresponding path also passes through node $n - 1$).

It is easy to see that we have an answer whether or not the Hamiltonian Path Problem for γ has a solution after at most $n^2 - n - 1$ steps performed by the P system Π_γ . This conclusion deserves to be stated as a theorem:

Theorem 7. *The Hamiltonian Path Problem can be solved by P systems with worm-objects in a quadratic time with respect to the number of nodes.*

The two phases of our computation, generating all candidate paths (in membrane 0) and then checking whether or not at least a path is Hamiltonian (in membranes $2, 3, \dots, n - 1$), are similar to the two phases of Adleman's algorithm (Adleman, 1994), with the difference that we need a quadratic time for producing the candidate solutions (Adleman performs this in a constant parallel biochemical time); however, we grow only candidate solutions of a prescribed length.

Now, we can proceed as in (Lipton, 1995), extending the previous procedure to the SAT problem. Because the truth-assignments to n variables x_1, \dots, x_n correspond to the paths from a given node to another given node in a graph with outdegree 2 (see Lipton, 1995), the solution is obtained in linear time: n steps for generating all truths-assignments and m steps to check the truth value of each of the m clauses.

A linear time solution to SAT can be also obtained by using P systems with symbol-objects (hence using multisets) and with the possibility to divide membranes, [P35: Gh. Păun, 2001]:

one works with polarized membranes; all rules are associated with membranes (this corresponds to the activity of enzymatic proteins embedded in bio-membranes), in the sense that the membranes themselves are part of evolution rules; under the influence of objects, the membranes can be divided and by such an operation an existing membrane is replaced by two copies of it (also the contents of the former membrane is copied in the new membranes); a membrane can also be divided if it contains two membranes of opposite polarization. The obtained systems are again computationally complete. Moreover, by using the membrane division, the number of membranes can increase exponentially and in this way we can solve SAT in linear time. We do not formally introduce these variants of P systems with active membranes because of the lack of space (both the definitions and the proofs are rather complex, at least as it concerns the formalization).

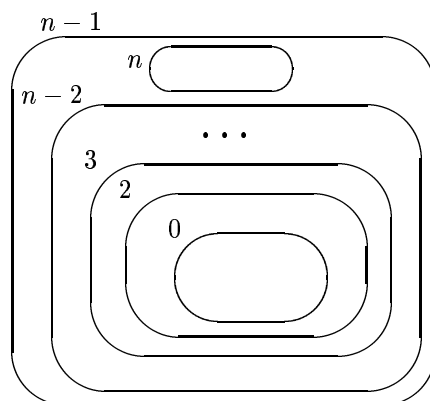


Figure 9: The membrane structure for solving HPP

A slight generalization was introduced in [P18: Krishna, Rama, 1999]: a membrane can be divided into an arbitrary number of copies, not only into two, as above. Such systems were shown to be able to solve the HPP and the Node Covering Problem, as well as to break the Data Encryption Standard (DES) in linear time. Of course, we always use an exponential space, which is a “detail” of a clear practical significance.

In [P35: Gh. Păun, 2001] and [P18: Krishna, Rama, 1999] one uses both the dissolving of elementary membranes, under the influence of objects present in them, and of non-elementary membranes, under the influence of membranes of opposite polarizations present in them. The second feature is rather unrealistic from a biological point of view, but it can be avoided without losing the computational universality, [P14: Freund, Martin-Vide, Gh. Păun, 2000], [P30: A. Păun, 2000]. It can also be avoided when solving NP-complete problems in polynomial (actually, linear) time providing that cooperative rules are used. This is shown in [P41: Gh. Păun, Suzuki, Tanaka, 2000] for ten problems, five from logic and five from graph theory. What is quite interesting and appealing from a practical point of view is that the P systems for all these problems have a very similar shape, namely, as suggested in Figure 10.

The general structure (and functioning) of these systems is as follows:

1. Always we start with two membranes, always the central one is divided into an exponential number of copies.
2. In a central “parallel engine” one generates, making use of the membrane division, a “data pool” of an exponential size; due to the parallelism, this takes, however, only a linear time. In parallel with this process, a “timer” is simultaneously tick-ing, in general, for synchronization reasons.

3. After finishing the generation of the “data pool”, one checks whether or not any solution to our problem exists; this is the step where we need cooperative rules.
4. A message is sent out of the system at a precise moment telling whether or not the problem has a solution. In all cases, the last two steps are done in a constant number of time units, again making use of the parallelism.

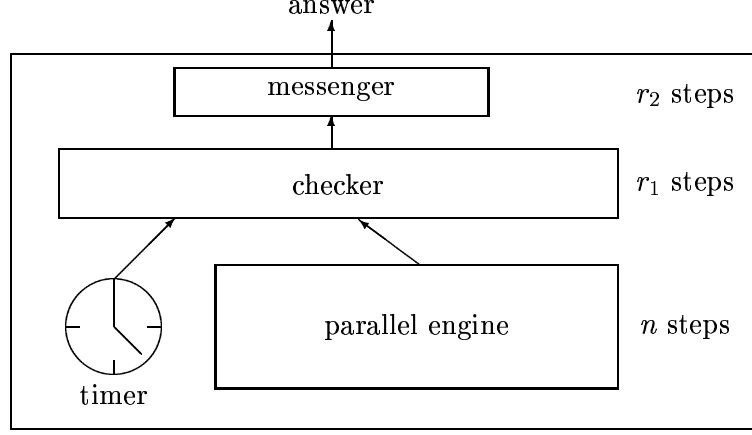


Figure 10: The shape of P systems solving NP-complete problems

For the reader convenience, we recall from [P41: Gh. Păun, Suzuki, Tanaka, 2000] the case of the SAT problem.

INSTANCE: A collection C of clauses $C_i = y_{i,1} \vee \dots \vee y_{i,p_i}$, $1 \leq i \leq m$, for some $m \geq 1, p_i \geq 1$, and $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$, $n \geq 1$, for each $1 \leq i \leq m, 1 \leq j \leq p_i$.

QUESTION: Is there a satisfying truth-assignment for C ?

P SYSTEM:

$$\begin{aligned}
V &= \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{c_i \mid 1 \leq i \leq n+1\} \cup \{k_i \mid 1 \leq i \leq m\} \cup \{yes\}, \\
\mu &= [{}_1[{}_0 \]_0]_1, \\
w_0 &= c_1 a_1 \dots a_n, \\
R &= \{[{}_0 a_i]_0 \rightarrow [{}_0 t_i]_0 [{}_0 f_i]_0 \mid 1 \leq i \leq n\} \text{ (parallel engine),} \\
&\cup \{[{}_0 c_i \rightarrow c_{i+1}]_0 \mid 1 \leq i \leq n\} \text{ (timer)} \\
&\cup \{[{}_0 c_{n+1} t_i \rightarrow k_1 t_i]_0 \mid x_i \text{ appears in } C_1, 1 \leq i \leq n\} \\
&\cup \{[{}_0 c_{n+1} f_i \rightarrow k_1 f_i]_0 \mid \neg x_i \text{ appears in } C_1, 1 \leq i \leq n\} \\
&\cup \{[{}_0 k_j t_i \rightarrow k_{j+1} t_i]_0 \mid x_i \text{ appears in } C_{j+1}, 1 \leq i \leq n, 1 \leq j \leq m-1\} \\
&\cup \{[{}_0 k_j f_i \rightarrow k_{j+1} f_i]_0 \mid \neg x_i \text{ appears in } C_{j+1}, 1 \leq i \leq n, 1 \leq j \leq m-1\} \\
&\quad \text{(truth-checking rules)} \\
&\cup \{[{}_0 k_m]_0 \rightarrow [{}_0 \]_0 yes, [{}_1 yes]_1 \rightarrow [{}_1 \]_1 yes\} \text{ (messenger rules)}.
\end{aligned}$$

In n steps, we generate all 2^n truths-assignments of the n variables (a rule $[{}_0 a_i]_0 \rightarrow [{}_0 t_i]_0 [{}_0 f_i]_0$ divides the membrane with label 0; the object a_i is replaced in the obtained membranes by t_i, f_i , respectively, indicating the *true* and *false* values of variable x_i); at the same time, the counter c_i arrives at c_{n+1} and starts checking the truth of clauses; if a clause C_j is valid, then the object

k_j is introduced. Checking all clauses takes m steps. (The checking rules are “localized”, they are applied in the 2^n copies of the membrane with label 0; this is indicated by writing the rules in the form $[_0\alpha \rightarrow \beta]_0$.) If in any copy of the membrane 0 we can obtain the object k_m , then the “message” *yes* is sent to membrane 1 and from here out of the system. This means that the set C of clauses is satisfiable if and only if at step $n + m + 2$ we get the object *yes* outside the system.

Note that for each clause we have as many checking rules as many literals we have in the clause and that the cooperative rules always involve two objects in their left hand sides.

11 Further Results, Research Topics

There are many results in P systems area, other than referring to characterizations of the power of Turing machines and to solving hard problems in a feasible (parallel) time. The bibliography which follows is meant to help the reader in having an image about the current state of the domain. We only mention here some of the main problems approached and results given in some of these papers: bridge theorems, relating the internal and the external modes of reading the result of a computation ([P38: Gh. Păun, Rozenberg, Salomaa, 2000]); further comparisons with (Parikh images of) languages in families from Chomsky and Lindenmayer hierarchies ([P8: Dassow, Gh. Păun, 1999], [P38: Gh. Păun, Rozenberg, Salomaa, 2000]); a characterization of Parikh sets of ETOL languages, by means of P systems with symbol-objects and with the possibility of creating new membranes by rules of the form $a \rightarrow [_i v]_i$, where $a \rightarrow v$ is a usual rule and i is the label of a membrane ([P16: Ito, Martin-Vide, Gh. Păun, 2000]); decidability results, in particular, whether or not a given membrane is ever dissolved during a computation ([P38: Gh. Păun, Rozenberg, Salomaa, 2000]); normal forms of membrane structures ([P39: Gh. Păun, Sakakibara, Yokomori, 1999], [P45: Petre, 1999]) and of the sets of rules present in each membrane ([P53: Zandron, Feretti, Mauri, 2000c]); rewriting P systems with a parallel use of rules, in the sense of L systems ([P19: Krishna, Rama, 2000a]); the power of synchronization ([P44: Gh. Păun, Yu, 1999]) and of communication ([P29: Martin-Vide, Gh. Păun, Rozenberg, 2000]); considering the possibility that also the rules can be moved from a membrane to another one ([P10: Freund, 1999]); P systems versus Cardelli’s ambient calculus ([P46: Petre, Petre, 1999]); implementations in Prolog ([P24: Malița, 2000]) and Lisp ([P48: Suzuki, Tanaka, 2000]); P systems with string-objects processed by means of cut-and-paste operations, related to the splicing operation ([P11: Freund, 1999a]); variants of P systems with active membranes ([P2: Atanasiu, 2000a], [P18: Krishna, Rama 1999]), either more restrictive or more general than those considered in [P37: Gh. Păun, 2001]; P systems whose rules produce or consume energy (at each step, in each membrane, the total energy should be positive, but if it exceeds a given threshold, then the membrane is dissolved ([P40: Gh. Păun, Suzuki, Tanaka, 2000]); P systems with picture objects ([P23: Krishna, Krithivasan, Rama, 2000]); P systems with active membranes which can generate context-free string languages by encoding the ordering of symbols from a string in the ordering of membranes ([P3: Atanasiu, Martin-Vide, 2000]), etc, etc.

The current bibliography of the domain, news, open problems, several papers, addresses of people working in the area can be found at the web page <http://bioinformatics.bio.disco.unimib.it/psystems>.

On the other hand, many open problems and research topics wait for further investigations. Technical open problems can be found in most of the papers available up to now, while [P36: Gh. Păun, 2000] is entirely devoted to listing some research directions of interest in this moment. We only recall some of them: consider reversible systems, deterministic systems, take care of the matter conservation law when devising the rules, look for hierarchies on the number

of membranes and on the depth of the membrane structure, find counterexamples able to separate classes of languages and of sets of vectors of numbers computed by P systems, consider probabilities, fuzzy sets and rough sets approaches, introduce some adaptability/learnability features (like in Neural Networks), look for further NP-complete problems which can be solved in polynomial time, and so on and so forth. Of course, the main research topic concerns the implementation of P systems, in bio-media or on an electronic support, but this, we have also said it before, is a long range interdisciplinary task.

Bibliography of P systems (October 2000)

- [P1] Atanasiu, A., 2000. Arithmetic with membranes. Pre-proc. Workshop on Multiset Processing, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 1–17.
- [P2] Atanasiu, A., Martin-Vide, C., 2000a. Recursive calculus with membranes.
- [P3] Atanasiu, A., Martin-Vide, C., 2000b. P systems and context-free languages.
- [P4] Baranda, A.V., Castellanos, J., Molina, R., Arroyo, F., Mingo, L.F., 2000. Data structures for implementing transition P systems in silico. Pre-proc. Workshop on Multiset Processing, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 21–34.
- [P5] Calude, C., Păun, Gh., 2000., Computing with Cells and Atoms. Taylor and Francis, London (Chapter 3: “Computing with Membranes”).
- [P6] Castellanos, J., Păun, Gh., Rodriguez-Paton, A., 2000. P systems with worm-objects. IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000, La Coruna, Spain, 64–74.
- [P7] Ciobanu, G., Paraschiv, D., 2000. Membrane software. A P system simulator.
- [P8] Dassow, J., Păun, Gh., 1999. On the power of membrane computing. J. of Universal Computer Sci., 5, 2, 33–49 (www.iicm.edu/jucs).
- [P9] Dassow, J., Păun, Gh., 2000. Concentration controlled P systems. Submitted.
- [P10] Freund, R., 1999a. Generalized P systems. Fundamentals of Computation Theory, FCT’99, Iaşi, 1999, (G. Ciobanu, Gh. Păun, eds.), LNCS 1684, Springer, 281–292.
- [P11] Freund, R., 1999b. Generalized P systems with splicing and cutting/recombination. Grammars, 2, 3, 189–199.
- [P12] Freund, R., 2000. Sequential P systems. Workshop on New Computing Paradigms, TU University Vienna, 177–183.
- [P13] Freund, R., Freund, F., 2000. Molecular computing with generalized homogeneous P systems. Proc. Conf. DNA6 (A. Condon, G. Rozenberg, eds.), Leiden, 113–125.
- [P14] Freund, R., Martin-Vide, C., Păun, Gh., 2000. Computing with membranes: Three more collapsing hierarchies. Submitted.
- [P15] Frisco, P., 2000. Membrane computing based on splicing: improvements. Pre-proc. Workshop on Multiset Processing, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 100–111.
- [P16] Ito, M., Martin-Vide, C., Păun, Gh., 2000. A characterization of Parikh sets of ETOL languages in terms of P systems. Submitted.
- [P17] Krishna, S.N., 2000. Computing with simple P systems. Pre-proc. Workshop on Multiset Processing, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 124–137.
- [P18] Krishna, S.N., Rama, R., 2000. A variant of P systems with active membranes: Solving NP-complete problems. Romanian J. of Information Science and Technology, 2, 4, 357–367.
- [P19] Krishna, S.N., Rama, R., 2000a. On the power of P systems with sequential and parallel rewriting. Intern. J. Computer Math., 77, 1-2, 1–14.
- [P20] Krishna, S.N., Rama, R., 2000b. Computability, complexity and languages of P systems. Submitted.
- [P21] Krishna, S.N., Rama, R., 2000c. On simple P systems with external output. Submitted.
- [P22] Krishna, S.N., Rama, R., 2000d. A note on partial/full parallel rewriting in P systems. Submitted.
- [P23] Krishna, S.N., Krithivasan, K., Rama, R., 2000. P systems with picture objects. Submitted.

- [P24] Malița, M., 2000. Membrane computing in Prolog. Pre-proc. Workshop on Multiset Processing, Curtea de Argeș, Romania, TR 140, CDMTCS, Univ. Auckland, 159–175.
- [P25] Manca, V., 2000. Monoidal systems and membrane systems. Pre-proc. Workshop on Multiset Processing, Curtea de Argeș, Romania, TR 140, CDMTCS, Univ. Auckland, 176–190.
- [P26] Martin-Vide, C., Mitrana, V., 2000. P systems with valuations. Proc. Second Conf. Unconventional Models of Computing (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag.
- [P27] Martin-Vide, C., Păun, Gh., 2000a. Computing with membranes. One more collapsing hierarchy. Bulletin of the EATCS, 72.
- [P28] Martin-Vide, C., Păun, Gh., 2000b. String objects in P systems. Proc. of Algebraic Systems, Formal Languages and Computations Workshop, Kyoto, 2000, RIMS Kokyuroku, Kyoto Univ.
- [P29] Martin-Vide, C., Păun, Gh., Rozenberg, G., 2000. Membrane systems with carriers. Submitted.
- [P30] Păun, A., 2000. On P systems with membrane division. Proc. Second Conf. Unconventional Models of Computing (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag.
- [P31] Păun, A., Păun, M., 2000. On the membrane computing based on splicing. Words, Languages, Grammars (C. Martin-Vide, V. Mitrana, eds.), Kluwer, Dordrecht.
- [P32] Păun, Gh., 1998. Computing with membranes, Journal of Computer and System Sciences, 61, 1 (2000), 108-143 (paper circulated since November 1998 as Turku Center for Computer Science-TUCS Report No 208, www.tucs.fi).
- [P33] Păun, Gh., 1999. Computing with membranes. An introduction. Bulletin of the EATCS, 67, 139–152.
- [P34] Păun, Gh., 2000a. Computing with membranes – A variant: P systems with polarized membranes. Intern. J. of Foundations of Computer Science, 11, 1, 167–182.
- [P35] Păun, Gh., 2000b. Computing with membranes; Attacking NP-complete problems. Proc. Second Conf. Unconventional Models of Computing (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag.
- [P36] Păun, Gh., 2000c. Computing with membranes (P systems): Twenty six research topics. Auckland University, CDMTCS Report No 119 (www.cs.auckland.ac.nz/CDMTCS).
- [P37] Păun, Gh., 2001. P systems with active membranes: Attacking NP-complete problems. J. Automata, Languages and Combinatorics, 6, 1.
- [P38] Păun, Gh., Rozenberg, G., Salomaa, A., 2000. Membrane computing with external output. Fundamenta Informaticae, 41, 3, 259–266.
- [P39] Păun, Gh., Sakakibara, Y., Yokomori, T., 1999. P systems on graphs of restricted forms. Submitted.
- [P40] Păun, Gh., Suzuki, Y., Tanaka, H., 2000. P Systems with energy accounting. Submitted.
- [P41] Păun, Gh., Suzuki, Y., Tanaka, H., Yokomori, T., 2000. On the power of membrane division in P systems. Submitted.
- [P42] Păun, Gh., Yokomori, T., 1999. Membrane computing based on splicing. Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers (E. Winfree, D. Gifford, eds.), MIT, 213–227.
- [P43] Păun, Gh., Yokomori, T., 2000. Simulating H systems by P systems. Journal of Universal Computer Science, 6, 1, 178–193 (www.iicm.edu/jucs).
- [P44] Păun, Gh., Yu, S., 1999. On synchronization in P systems. Fundamenta Informaticae, 38, 4, 397–410.
- [P45] Petre, I., 1999. A normal form for P systems. Bulletin of the EATCS, 67, 165–172.
- [P46] Petre, I., Petre, L., 1999. Mobile ambients and P systems. J. Universal Computer Sci., 5, 9, 588–598.
- [P47] Rama, R., 2000. Computing with P systems. Pre-proc. Workshop on Multiset Processing, Curtea de Argeș, Romania, TR 140, CDMTCS, Univ. Auckland, 218–235.
- [P48] Suzuki, Y., Tanaka, H., 2000a. On a LISP implementation of a class of P systems. Romanian J. of Information Science and Technology, 3, 2.
- [P49] Suzuki, Y., Tanaka, H., 2000b. Artificial life and P systems. Pre-proc. Workshop on Multiset Processing, Curtea de Argeș, Romania, TR 140, CDMTCS, Univ. Auckland, 265–285.
- [P50] Ștefan, Gh., 2000. Membrane computation with cellular automata, on a dedicated hardware, Workshop on Multiset Processing, Curtea de Argeș, Romania.

- [P51] Zandron, Cl., Ferretti, Cl., Mauri, G., 2000a. Two normal forms for rewriting P systems. Submitted.
- [P52] Zandron, Cl., Ferretti, Cl., Mauri, G., 2000b. Priorities and variable thickness of membranes in rewriting P systems. Submitted.
- [P53] Zandron, Cl., Ferretti, Cl., Mauri, G., 2000c. Universality and normal forms on membrane systems. Proc. Intern. Workshop Grammar Systems 2000 (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, 61–74.
- [P54] Zandron, Cl., Ferretti, Cl., Mauri, G., 2000d. Solving NP-complete problems using P systems with active membranes. Proc. Second Conf. Unconventional Models of Computing (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag.
- [P55] Zandron, Cl., Ferretti, Cl., Mauri, G., 2000e. Using membrane features in P systems. Preproc. Workshop on Multiset Processing, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 296–320.

References

- [1] Adleman, L.M., 1994. Molecular computation of solutions to combinatorial problems. *Science*, 226, 1021–1024.
- [2] Alberts, B., et al., 1998. *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*. Garland Publ. Inc., New York, London.
- [3] Bray, D., 1995. Protein molecules as computational elements in living cells. *Nature*, 376, 307–312
- [4] Dassow, J., Păun, Gh., 1989. *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin.
- [5] Freund, R., 2000. On the number of variables in graph-grammars, programmed, and matrix grammars. Submitted.
- [6] Head, T., 1987. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49, 737–759.
- [7] Lipton, R.J., 1995. Using DNA to solve NP-complete problems. *Science*, 268, 542–545.
- [8] Loewenstein, W.R., 1999. *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*. Oxford Univ. Press, New York, Oxford.
- [9] Mader, S.S., 1996. *Biology (Fifth Edition)*. McGraw-Hill, Boston (Chapter 6: Membrane structure and function, 84–102).
- [10] Păun, Gh., 1984. Six nonterminals are enough for generating each r.e. language by a matrix grammar. *Intern. J. Computer Math.*, 15, 23–37.
- [11] Păun, Gh., Rozenberg, G., Salomaa, A., 1998. *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin.
- [12] Rozenberg, G., Salomaa, A., eds., 1997. *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin.
- [13] Sipper, M., 1995. Studying Artificial Life using a simple, general cellular model. *Artificial Life Journal*, 2, 1, 1–35.