Chapter 1
Introduction to Membrane Computing

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
`george.paun@imar.ro`

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
`gpaun@us.es`

Summary. This is a comprehensive (and friendly) introduction to membrane computing (MC), meant to offer both computer scientists and non-computer scientists an up-to-date overview of the field. That is why the set of notions introduced here is rather large, but the presentation is informal, without proofs and with rigorous definitions given only for the basic types of P systems – symbol object P systems with multiset rewriting rules, systems with symport/antiport rules, systems with string objects, tissue-like P systems, and neural-like P systems. Besides a list of (biologically inspired or mathematically motivated) ingredients/features which can be used in systems of these types, we also mention a series of results, as well as a series of research trends and topics.

1 (The Impossibility of) A Definition of Membrane Computing

Membrane computing (MC) is an area of computer science aiming to abstract computing ideas and models from the structure and the functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures.

In short, MC deals with distributed and parallel computing models, processing multisets of symbol objects in a localized manner (evolution rules and evolving objects are encapsulated into compartments delimited by membranes), with an essential role played by the communication between compartments (and with the environment). Of course, this is just a rough description of a membrane system – hereafter called P system – of the very basic type, as many different classes of such devices exist.

The essential ingredient of a P system is its *membrane structure*, which can be a hierarchical arrangement of membranes, as in a cell (hence described by a tree), or a net of membranes (placed in the nodes of a graph), as in a tissue or a neural net. The intuition behind the notion of a membrane is a three-dimensional vesicle from biology, but the concept itself is generalized/idealized to interpreting a membrane as a *separator* of two regions (of Euclidean space), a finite "inside" and an infinite "outside," providing the possibility of *selective communication* between the two regions.

The variety of suggestions from biology and the range of possibilities to define the architecture and the functioning of a membrane-based multiset processing device are practically endless, and already the literature of MC contains a very large number of models. Thus, MC is not merely a theory related to a specific model, it is a *framework* for devising compartmentalized models. Because the domain is rather young (the trigger paper is [64], circulated first on the Web, though related ideas were considered before, in various contexts), and as a genuine feature, based on both the biological background and the mathematical formalism used, not only are there already many types of proposed P systems, but also the flexibility and the versatility of P systems seem, in principle, to be unlimited.

This last observation, as well as the rapid development and enlargement of the research in this area, make impossible a short and faithful presentation of membrane computing.

However, there are series of notions, notations, and models which are already "standard," which have stabilized and can be considered as basic elements of MC. This chapter is devoted to presenting mainly such notions and models, together with their notations.

The presentation will be both historically and didactically organized, introducing mainly notions first investigated in this area, or simple notions able to quickly offer an idea of membrane computing to the reader not familiar with the domain.

The reader has surely noticed that the discussion refers mainly to computer science (goals), and much less to biology. MC was not initiated as an area aiming to provide models to biology, models of the cell in particular. At this moment, after considerable development at the theoretical level, the domain is not yet fully prepared to offer such models to biology, though this has been an important direction of the recent research, and considerable advances toward such achievements have been reported. The present volume is a proof of this assertion.

2 Membrane Computing as Part of Natural Computing

Before entering into more specific elements of MC, let us spend some time with the relationship of this area with, let us say, the "local" terminology, the "outside." We have said above that MC is part of computer science. However, the *genus proximus* is natural computing, the general attempt to learn ideas, models, and paradigms useful to computer science from the way nature – life, especially – "computes" in various circumstances where substance and information processing can be interpreted as computation. Classic bio-inspired branches of natural computing are genetic algorithms (more generally, evolutionary computing) and neural computing. Both have long histories, which can be traced to the unpublished works of Turing, many applications, and a huge bibliography. Both are proof that "it is worth learning from biology," supporting the optimistic observation that during many billions of years nature/life has adjusted certain tools and processes which, correctly abstracted and implemented in computer science terms, can prove to be surprisingly useful in many applications.

A more recent branch of natural computing, with an enthusiastic beginning and as yet unconfirmed computational applicability (we do not discuss here the by-products, such as the nanotechnology related developments), is DNA computing, whose birth is related to the Adleman experiment [1] of solving a (small) instance of the Hamiltonian path problem by handling DNA molecules in a laboratory. According to Hartmanis [39, 40], this was a *demo* that we can compute with biomolecules, a big event for computability. However, after one decade of research, the domain is still preparing its tools for a possible future practical application and looking for a new breakthrough idea, similar to Adleman's one from 1994.

Both evolutionary computing and DNA computing are inspired from and related to DNA molecules. Neural computing considers the neurons as simple finite automata linked in specific types of networks. Thus, these "neurons" are not interpreted as cells, with an internal structure and life, but as "dots on a grid", with a simple input-output function. (The same observation holds true for cellular automata, where again the "cells" are "dots on a grid," interacting only among themselves, in a rigid structure.) None of these domains considers the cell itself as its main object of research; in particular, none of these domains pays any attention to membranes and compartmentalization – and this is the point where membrane computing enters the stage. Thus, MC can be seen as an extension of DNA (or, more generally, molecular) computing, from the "one processor" level to a distributed computing model.

3 Laudation to the Cell (and Its Membranes)

Life (as we know it on earth in the traditional meaning of the term, that investigated by biology) is directly related to cells; everything alive consists of cells or has to do in a direct way with cells. The cell is the smallest "thing" unanimously considered *alive*. It is very small and very intricate in its structure and functioning, has elaborate internal activity and complex interaction with the neighboring cells and with the environment. It is fragile and robust at the same time, with a way to organize (control) the biochemical (and informational) processes developed during billions of years of evolution.

Cell means membranes. The cell itself is defined – separated from its environment – by a membrane, the external one. Inside the cell, several membranes enclose "protected reactors," compartments where specific biochemical processes take place. In particular, a membrane encloses the nucleus (of eukaryotic cells), where the genetic material is placed. Through vesicles enclosed by membranes one can transport packages of molecules from a part of the cell (e.g., from the Golgi apparatus) to other parts of the cell in such a way that the transported molecules are not "available" during their journey to neighboring chemicals.

The membranes allow a selective passage of substances between the compartments delimited by them. This can be a simple selection by size in the case of small molecules, or a much more intricate selection, through protein channels which do not only select but can also move molecules from a low concentration to a higher concentration, perhaps coupling molecules, through so-called symport and antiport processes.

Moreover, the membranes of a cell do not delimit only compartments where specific reactions take place in solution, *inside* the compartments, but many reactions in a cell develop *on the membranes*, catalyzed by the many proteins bound to them. It is said that when a compartment is too large for the local biochemistry to be efficient, life creates membranes, both in order to create smaller "reactors" (small enough that, through the Brownian motion, any two of the enclosed molecules can collide – hence, react – frequently enough) and in order to create further "reaction surfaces." Anyway, biology contains many fascinating facts from a computer science point of view, and the reader is encouraged to check the validity of this assertion, e.g., through [2, 53, 7].

Life means surfaces inside surfaces, as can be learned from the title of [41], while S. Marcus puts it in an equational form [56]: *Life = DNA software + membrane hardware.*

There are cells living alone (unicellular organisms, such as ciliates, bacteria, etc.), but in general the cells are organized as tissues, organs, organisms, and communities of organisms. All these suppose a specific organization, starting with the direct communication/cooperation among neighboring cells and ending with the interaction with the environment at various levels. Together with the internal structure and organization of the cell, these suggest a lot of ideas, exciting from a mathematical point of view, and potentially useful from a computability point of view. Some of them have already been explored in MC, but many more still await research efforts (for example, the brain, the best "computer" ever invented, is still a major challenge for mathematical modeling).

4 Some General Features of Membrane Computing Models

It is worth mentioning from the beginning, besides the essential use of membranes/compartmentalization, some of the basic features of models investigated in this field.

We have mentioned above the notion of a multiset. The compartments of a cell contain substances (ions, small molecules, macromolecules) swimming in an aqueous solution. There is no ordering there; everything is close to everything; the concentration matters, i.e., the population, *the number of copies of each molecule* (of course, we are abstracting/idealizing here, departing from the biological reality). Thus, the suggestion is immediate: to work with sets of objects whose multiplicities matters; hence, with *multisets.* This is a data structure with peculiar characteristics, not new but not systematically investigated in computer science.

A multiset can be represented in many ways, but the most compact one is in the form of a string. For instance, if the objects a, b, and c are present, respectively, in 5, 2, and 6 copies each, we can represent this multiset by the string $a^5b^2c^6$; of course, all permutations of this string represent the same multiset.

The string representation of multisets and the biochemical background, where standard chemical reactions are common, suggest processing the multisets from the compartments of our computing device by means of rewriting-like rules; this means rules of the form $u \to v$, where u and v are multisets of objects (represented by strings). Continuing the previous example, we can consider a rule $aab \to abcc$. It indicates that two copies of object a and a copy of object b react, and, as a result of this reaction, we get back a copy of a as well as the copy of b (hence b behaves here as a catalyst), and we produce two new copies of c. If this rule is applied to the multiset $a^5b^2c^6$, then, because aab are "consumed" and then $abcc$ are "produced," we obtain the multiset $a^4b^2c^8$. Similarly, by using the rule $bb \to aac$, we get the multiset a^7c^7, which contains no occurrence of object b.

Two important problems arise here. The first one is related to the *nondeterminism.* Which rules should be applied and to which objects? The copies of an object are considered identical, so we do not distinguish among them; whether to use the first rule or the second one is a significant issue, especially because they cannot be both used at the same time (for the multiset mentioned), as they compete for the "reactant" b. The standard solution to this problem in membrane computing is that *the rules and the objects are chosen in a nondeterministic manner* (at random, with no preference; more rigorously, we can say that any possible evolution is allowed).

This is also related to the idea of *parallelism.* Biochemistry is not only (to a certain degree) nondeterministic, but it is also (to a certain degree) parallel. If two chemicals can react, then the reaction does not take place for only two molecules of the two chemicals, but, in principle, for all molecules. This is the suggestion supporting the maximal parallelism used in many classes of P systems: at each step, all rules which can be applied have to be applied to all possible objects. We will come back to this important notion later, but now we illustrate it only with the previous multiset and pair of rules. Using these rules in the maximally parallel manner means either using the first rule twice (thus involving four copies of a and both copies of b) or using the second rule once (it consumes both copies of b, hence the first rule cannot be used at the same time). In the first case, one copy of a remains unused (and the same for all copies of c), and the resulting multiset is $a^3b^2c^{10}$; in the second case, all copies of a and c remain unused, and the resulting multiset is a^7c^7. Note that in the latter case the maximally parallel application of rules corresponds to the *sequential* (one in a time) application of the second rule.

There are also other types of rules used in MC (e.g., symport and antiport rules), but we will discuss them later. Here we conclude with the observation that MC deals with models which are intrinsically *discrete* (basically, working with multisets of objects, with the multiplicities being natural numbers) and evolve through *rewriting-like* (we can also say reaction-like) rules.

5 Computer Science Related Areas

Rewriting rules are standard rules for handling strings in formal language theory (although other types of rules, such as insertion, deletion, context-adjoining, are also used both in formal language theory and in P systems). Similarly, working with strings modulo the ordering of symbols is another old idea: commutative languages (investigated, e.g., in [28]) are nothing other than the permutation closure of languages. In turn, the multiplicity of symbol occurrences in a string corresponds to the Parikh image of the string, which directly leads to vector addition systems, Petri nets, register machines, and formal power series.

Parallelism is also considered in many areas of formal languages, and it is the main feature of Lindenmayer systems. These systems deserve a special discussion here, since they are a well developed branch of formal language theory inspired by biology, specifically, by the development of multi-cellular organisms (which can be described by strings of symbols). However, for L systems the cells are considered as symbols; their organization in (mainly linear) patterns, not their structure, is investigated. P systems can be seen as dual to L systems, as they zoom in the cell, distinguishing the internal structure and the objects evolving inside it, maybe also distinguishing (when "zooming enough") the structure of the objects, which leads to the category of P systems with string objects.

However, a difference exists between the kind of parallelism in L systems and that in P systems: in L systems the parallelism is *total* – all symbols of a string are processed at the same time; in P systems we work with a maximal parallelism – we process as many objects as possible, but not necessarily all of them.

Still closer to MC are the multiset processing languages, the most known of them being Gamma [8, 9]. The standard rules of Gamma are of the form $u \rightarrow v(\pi)$, where u and v are multisets and π is a predicate which should be satisfied by the multiset to which the rule $u \rightarrow v$ is applied. The generality of the form of rules ensures great expressivity and, in a direct manner, computational universality. What Gamma does not have (at least in the initial versions) is distributivity. Then, MC restricts the form of rules, on the one hand as imposed by the biological roots and on the other hand in search of mathematically simple and elegant models.

Membranes appear even in Gamma-related models, and this is the case with CHAM, the Chemical Abstract Machine of Berry and Boudol, [12], the direct ancestor of membrane systems; however, the membranes of CHAM are not membranes as in cell biology, but correspond to the contents of membranes, i.e., multisets, and lower level membranes together, while the goals and the approach are completely different, directed to the algebraic treatment of the processes these membranes can undergo. From this point of view, of goals and tools, CHAM has a recent counterpart in the so-called *brane calculus* (of course, “brane” comes from “membrane”) from [17] (see also [74] for a related approach), where process algebra is used for investigating the processes taking place *on* membranes and *with* membranes of a cell.

The idea of designing a computing device based on compartmentalization through membranes was also suggested in [55].

Many related areas and many roots, with many common ideas and many differences! To some extent, MC is a synthesis of some of these ideas, integrated in a framework directly inspired by cell biology, paying deserved attention to membranes (and hence to distribution, hierarchization, communication, localization, and other related concepts), aiming – in the basic types of devices – to find computing models, as elegant (minimalistic) as possible, as powerful as possible (in comparison with Turing machines and their subclasses), and as efficient as possible (able to solve computationally hard problems in feasible time).

6 The Cell-Like Membrane Structure

We move now toward presenting in a more precise manner the computing models investigated in our area, and we start by introducing one the fundamental ingredients of a P system, namely, the *membrane structure*.

The meaning of this notion is illustrated in Figure 1, and this is what we can see when looking (through mathematical glasses, hence abstracting as much as necessary in order to obtain a formal model) at a standard cell.

Thus, as suggested by Figure 1, a membrane structure is a hierarchically arranged set of membranes, contained in a distinguished external membrane (corresponding to the plasma membrane and usually called the *skin* membrane). Several membranes can be placed inside the skin membrane (they correspond to the membranes present in a cell, around the nucleus, in Golgi apparatus, vesicles, mitochondria, etc.); a membrane without any other membrane inside it is said to be *elementary.* Each membrane determines a compartment, called *region*, the space delimited by it from above and from below by the membranes placed directly inside, if any exist. Clearly, the correspondence membrane-region is one-to-one; that is why we sometimes use the terms interchangeably.

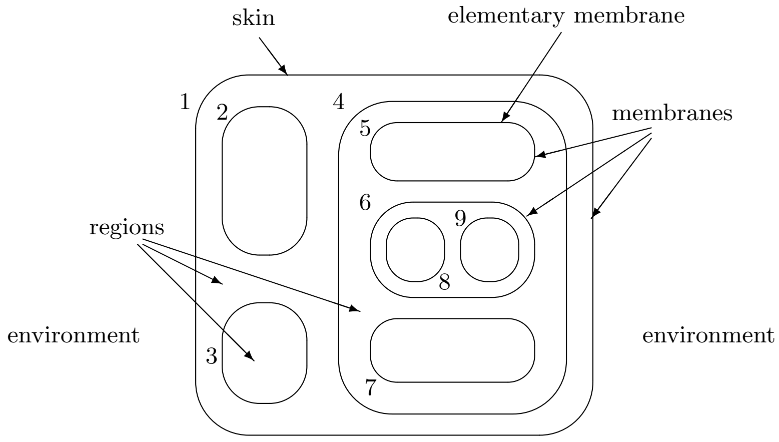


Fig. 1. A membrane structure.

Usually, the membranes are identified by *labels* from a given set of labels. In Figure 1, we use numbers, starting with number 1 assigned to the skin membrane (this is the standard labeling, but the labels can be more informative "names" associated with the membranes). Also, in the figure the labels are assigned in a one-to-one manner to membranes, but this is possible only in the case of membrane structures which cannot grow (indefinitely), otherwise several membranes would have the same label (we will later see such cases). Due to the membrane-region correspondence, we identify by the same label a membrane and its associated region.

Clearly, the hierarchical structure of membranes can be represented by a rooted tree; Figure 2 gives the tree which describes the membrane structure in Figure 1. The root of the tree is associated with the skin membrane and the leaves are associated with the elementary membranes. In this way, various graph-theoretic notions are brought onto the stage, such as the distance in the tree, the level of a membrane, the height/depth of the membrane structure, as well as terminology such as parent/child membrane, ancestor, etc.

Directly suggested by the tree representation is the symbolic representation of a membrane structure, by strings of labeled matching parentheses. For instance, a string corresponding to the structure from Figure 1 is the following:

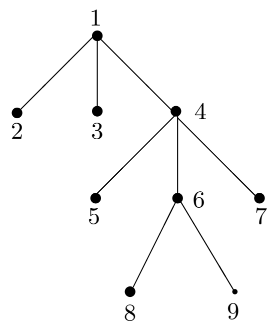


Fig. 2. The tree describing the membrane structure from Figure 1.

$$[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ [_8\ ]_8\ [_9\ ]_9\ ]_6\ [_7\ ]_7\ ]_4\ ]_1. \qquad (*)$$

An important aspect should now be noted: the membranes of the same level can float around, that is, the tree representing the membrane structure is not oriented; in terms of parentheses expressions, two subexpressions placed at the same level represent the same membrane structure. For instance, in the previous case, the expression

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

is a representation of the same membrane structure, equivalent to $(*)$.

7 Evolution Rules and the Way of Using Them

In the basic variant of P systems, each region contains a multiset of symbol objects, which correspond to the chemicals swimming in a solution in a cell compartment. These chemicals are considered here as unstructured; that is why we describe them with symbols from a given alphabet.

The objects evolve by means of *evolution rules*, which are also localized, associated with the regions of the membrane structure. Actually, there are three main types of rules: (1) multiset-rewriting rules (one calls them, simply, evolution rules), (2) communication rules, and (3) rules for handling membranes.

In this section we present the first type of rules. They correspond to the chemical reactions possible in the compartments of a cell; hence they are of the form $u \to v$, where u and v are multisets of objects. However, in order to make the compartments cooperate, we have to move objects across membranes, and for this we add *target indications* to the objects produced by a rule as above (to the objects from multiset v). These indications are *here, in,* and *out*, with the meanings that an object associated with the indication *here* remains in the same region, one associated with the indication *in* goes immediately into an adjacent lower membrane, nondeterministically chosen, and *out* indicates that the object has to exit the membrane, thus becoming an element of the region surrounding it. An example of an evolution rule is $aab \rightarrow (a, here)(b, out)(c, here)(c, in)$ (this is the first of the rules considered in Section 4, with target indications associated with the objects produced by rule application). After using this rule in a given region of a membrane structure, two copies of a and one of b are consumed (removed from the multiset of that region), and one copy of a, one of b, and two of c are produced; the resulting copy of a remains in the same region, and the same happens with one copy of c (indications *here*), while the new copy of b exits the membrane, going to the surrounding region (indication *out*), and one of the new copies of c enters one of the child membranes, nondeterministically chosen. If no such child membrane exists, that is, the membrane with which the rule is associated is elementary, then the indication *in* cannot be followed, and the rule cannot be applied. In turn, if the rule is applied in the skin region, then b will exit into the environment of the system (and it is "lost" there, since it can never come back). In general, the indication *here* is not specified (an object without an explicit target indication is supposed to remain in the same region where the rule is applied).

It is important to note that in this initial type of system we do not provide similar rules for the environment, since we do not care about the objects present there; later we will consider types of P systems where the environment also takes part in system evolution.

A rule such as the one above, with at least two objects in its left hand side, is said to be *cooperative*; a particular case is that of *catalytic* rules, of the form $ca \rightarrow cv$, where c is an object (called catalyst) which assists the object a to evolve into the multiset v; rules of the form $a \rightarrow v$, where a is an object, are called *non-cooperative.*

The rules can also have the form $u \rightarrow v\delta$, where δ denotes the action of *membrane dissolving*: if the rule is applied, then the corresponding membrane disappears and its contents, object and membranes alike, are left free in the surrounding membrane; the rules of the dissolved membrane disappear with the membrane. The skin membrane is never dissolved.

The communication of objects through membranes evokes the fact that biological membranes contain various (protein) channels through which the molecules can pass (in a passive way, due to concentration difference, or in an active way, with consumption of energy), in a rather selective manner. However, the fact that the communication of objects from a compartment to a neighboring compartment is controlled by the "reaction rules" is mathematically attractive, but is not quite realistic from a biological point of view; that is why variants were also considered where the two processes are separated: the evolution is controlled by rules as above, without target indications, and the communication is controlled by specific rules (e.g., by symport/antiport rules).

It is also worth noting that evolution rules are stated in terms of *names of objects*, while their application/execution is done using *copies of objects* – remember the example from Section 4, where the multiset $a^5b^2c^6$ was processed by a rule of the form $aab \to a(b, out)c(c, in)$, which, in the maximally parallel manner, is used twice, for the two possible sub-multisets aab.

We have arrived in this way at the important feature of P systems, concerning *the way of using the rules*. The key phrase in this respect is: *in the maximally parallel manner, nondeterministically choosing the rules and the objects*.

Specifically, this means that we assign objects to rules, nondeterministically choosing the objects and the rules until no further assignment is possible. Mathematically stated, we look to the *set* of rules, and try to find a *multiset* of rules, by assigning multiplicities to rules, with two properties: (i) the multiset of rules is *applicable* to the multiset of objects available in the respective region; that is, there are enough objects to apply the rules a number of times as indicated by their multiplicities; and (ii) the multiset is *maximal*, i.e., no further rule can be added to it (no multiplicity of a rule can be increased), because of the lack of available objects.

Thus, an evolution step in a given region consists of finding a maximal applicable multiset of rules, removing from the region all objects specified in the left hand sides of the chosen rules (with multiplicities as indicated by the rules and by the number of times each rule is used), producing the objects from the right hand sides of the rules, and then distributing these objects as indicated by the targets associated with them. If at least one of the rules introduces the dissolving action δ, then the membrane is dissolved, and its contents become part of the parent membrane, provided that this membrane was not dissolved at the same time; otherwise we stop at the first upper membrane which was not dissolved (the skin membrane at least remains intact).

8 A Formal Definition of a Transition P System

Systems based on multiset-rewriting rules as above are usually called *transition P systems*, and we preserve here this terminology (although "transitions" are present in all types of systems).

Of course, when presenting a P system we have to specify the alphabet of objects (a usual finite nonempty alphabet of abstract symbols identifying the objects), the membrane structure (it can be represented in many ways, but the one most used is by a string of labeled matching parentheses), the multisets of objects present in each region of the system (represented in the most compact way by strings of symbol objects), the sets of evolution rules associated with each region, and the indication about the way the output is defined (see below).

Formally, a *transition P system* (of degree $m \geq 1$) is a construct of the form

$$\Pi = (O, C, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i_o),$$

where:

1. O is the (finite and nonempty) alphabet of *objects*,
2. $C \subset O$ is the set of *catalysts*,
3. μ is a membrane structure, consisting of m membranes, labeled $1, 2, \ldots, m$; we say that the membrane structure, and hence the system, is *of degree* m,
4. $w_1, w_2, \ldots, w_m$ are strings over O representing the *multisets of objects* present in regions $1, 2, \ldots, m$ of the membrane structure,
5. $R_1, R_2, \ldots, R_m$ are finite *sets of evolution rules* associated with regions $1, 2, \ldots, m$ of the membrane structure,
6. i_o is either one of the labels $1, 2, \ldots, m$, and the respective region is the *output region* of the system, or it is 0, and the result of a computation is collected in the environment of the system.

The rules are of the form $u \to v$ or $u \to v\delta$, with $u \in O^+$ and $v \in (O \times Tar)^*$, where[1] $Tar = \{here, in, out\}$. The rules can be cooperative (with u arbitrary), non-cooperative (with $u \in O - C$), or catalytic (of the form $ca \to cv$ or $ca \to cv\delta$, with $a \in O - C, c \in C$, and $v \in ((O - C) \times Tar)^*$); note that the catalysts never evolve and never change the region, they only help the other objects to evolve.

A possible restriction about the region i_o in the case when it is an internal one is to consider only regions enclosed by elementary membranes for output (that is, i_o should be the label of an elementary membrane of μ).

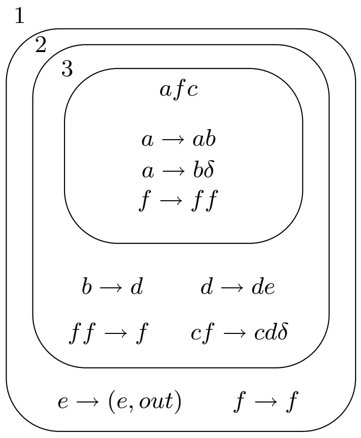


Fig. 3. The initial configuration of a P system, rules included.

In general, the membrane structure and the multisets of objects from its compartments identify a *configuration* of a P system. The *initial configuration*

[1] By V^* we denote the set of all strings over an alphabet V, the empty string λ included, and by V^+ we denote the set $V^* - \{\lambda\}$ of all nonempty strings over V.

is given by specifying the membrane structure and the multisets of objects available in its compartments at the beginning of a computation, that is, by $(\mu, w_1, \ldots, w_m)$. During the evolution of the system, by applying the rules, both the multisets of objects and the membrane structure can change. We will see how this is done in the next section; here we conclude with an example of a P system, represented in Figure 3. It is important to note that adding the set of rules to the initial configuration, placed in the corresponding regions, we have a complete and concise presentation of the system (the indication of the output region can also be added in a suitable manner, for instance, writing "output" inside it).

9 Defining Computations and Results of Computations

In their basic variant, membrane systems are synchronous devices, in the sense that a global clock is assumed, which marks the time for all regions of the system. In each time unit a transformation of a configuration of the system – we call it *transition* – takes place by applying the rules in each region in a *nondeterministic* and *maximally parallel manner*. As explained in the previous sections, this means that the objects to evolve and the rules governing this evolution are chosen in a nondeterministic way, and this choice is "exhaustive" in the sense that, after the choice is made, no rule can be applied in the same evolution step to the remaining objects.

A sequence of transitions constitutes a *computation*. A computation is successful if it halts, reaches a configuration where no rule can be applied to the existing objects, and the output region i_o still exists in the halting configuration (in the case where i_o is the label of a membrane, it can be dissolved during the computation, but the computation is then no longer successful). With a successful computation we can associate a *result* in various ways. If we have an output region specified, and this is an internal region, then we have an *internal output*: we count the objects present in the output region in the halting configuration and this number is the result of the computation. When we have $i_o = 0$, we count the objects which leave the system during the computation, and this is called *external output*. In both cases the result is a number. If we distinguish among different objects, then we can have as the result a vector of natural numbers. The objects which leave the system can also be arranged in a sequence according to the time when they exit the skin membrane, and in this case the result is a string (if several objects exit at the same time, then all their permutations are accepted as a substring of the result). Note that non-halting computations provide no output (we cannot know when a number is "completely computed" before halting); if the output membrane is dissolved during the computation, then the computation aborts, and no result is obtained (of course, this makes sense only in the case of internal output).

A possible extension of the definition is to consider a *terminal* set of objects, $T \subseteq O$, and to count only the copies of objects from T, discarding the objects from $O - T$ present in the output region. This allows some additional leeway in constructing and "programming" a P system, because we can ignore some auxiliary objects (e.g., the catalysts).

Because of the nondeterminism of the application of rules, starting from an initial configuration we can get several successful computations, and hence several results. Thus, a P system *computes* (one also says *generates*) a set of numbers (or a set of vectors of numbers, or a language, depending on the way the output is defined). The case when we get a language is important in view of the qualitative difference between the "loose" data structure we use inside the system (vectors of numbers) and the data structure of the result, strings, where we also have a "syntax," a positional information.

For a given system Π we denote by $N(\Pi)$ the set of numbers computed by Π in the above way. When we consider the vector of multiplicities of objects from the output region, we write $Ps(\Pi)$. In turn, in the case where we take as (external) output the strings of objects leaving the system, we denote the language of these strings by $L(\Pi)$.

Let us illustrate the previous definitions by examining the computations of the system from Figure 3, with the output region being the environment.

We have objects only in the central membrane, that with label 3; hence only here can we apply rules. Specifically, we can repeatedly apply the rule $a \to ab$ in parallel with $f \to ff$, and in this way the number of copies of b grows each step by one, while the number of copies of f is doubled in each step. If we do not apply the rule $a \to b\delta$ (again in parallel with $f \to ff$), which dissolves the membrane, then we can continue in this way forever. Thus, in order to ever halt, we have to dissolve membrane 3. Assume that this happens after $n \geq 0$ steps of using the rules $a \to ab$ and $f \to ff$. When membrane 3 is dissolved, its contents ($n+1$ copies of b, 2^{n+1} copies of f, and one copy of the catalyst c) are left free in membrane 2, which can now start using its rules. In the next step, all objects b become d. Let us examine the rules $ff \to f$ and $cf \to cd\delta$. The second rule dissolves membrane 2, and hence passes its contents to membrane 1. If among the objects which arrive in membrane 1 there is at least one copy of f, then the rule $f \to f$ from region 1 can be used forever and the computation never stops; moreover, if the rule $ff \to f$ is used at least once, in parallel with the rule $cf \to cd\delta$, then at least one copy of f is present. Therefore, the rule $cf \to cd\delta$ should be used only if region 2 contains only one copy of f (note that, because of the catalyst, the rule $cf \to cd\delta$ can be used only for one copy of f). This means that the rule $ff \to f$ was used always for all available pairs of f, that is, at each step the number of copies of f is divided by 2. This is already done once in the step where all copies of b become d, and will be done from now on as long as at least two copies of f are present. Simultaneously, at each step, each d produces one copy of e. This process can continue until we get a configuration with only one copy of f present; at that step we have to use the rule $cf \to cd\delta$, hence also membrane 2 is dissolved. Because we have applied the rule $d \rightarrow de$, in parallel for all copies of d (there are $n+1$ such copies) during $n+1$ steps, we have $(n+1)(n+1)$ copies of e, $n+2$ copies of d (one of them produced by the rule $cf \rightarrow cd\delta$), and one copy of c in the skin membrane of the system (the unique membrane still present). The objects e are sent out, and the computation halts. Therefore, we compute in this way the number $(n+1)^2$ for $n \geq 0$, that is, $N(\Pi) = \{n^2 \mid n \geq 1\}$.

10 Using Symport and Antiport Rules

The multiset rewriting rules correspond to reactions taking place in the cell, *inside* the compartments. However, an important part of the cell activity is related to the passage of substances through membranes, and one of the most interesting ways to handle this trans-membrane communication is by coupling molecules. The process by which two molecules pass together across a membrane (through a specific protein channel) is called *symport*; when the two molecules pass simultaneously through a protein channel, but in opposite directions, the process is called *antiport*.

We can formalize these operations in an obvious way: (ab, in) or (ab, out) are symport rules, stating that a and b pass together through a membrane, entering in the former case and exiting in the latter case; similarly, $(a, out; b, in)$ is an antiport rule, stating that a exits and, at the same time, b enters the membrane. Separately, neither a nor b can cross a membrane unless we have a rule of the form (a, in) or (a, out), called, for uniformity, *uniport* rule.

Of course, we can generalize these types of rules, by considering symport rules of the form (x, in) and (x, out), and antiport rules of the form $(z, out; w, in)$, where x, z, and w are multisets of arbitrary size; one says that $|x|$ is the *weight* of the symport rule, and $\max(|z|, |w|)$ is the *weight* of the antiport rule[2].

Now, such rules can be used in a P system instead of the target indications *here, in*, and *out*: we consider multiset rewriting rules of the form $u \rightarrow v$ (or $u \rightarrow v\delta$) without target indications associated with the objects from v, as well as symport/antiport rules for communication of the objects between compartments. Such systems, called *evolution-communication* P systems, were considered in [18] (for various restricted types of rules of the two forms).

Here, we do not go into that direction, but stay closer both to the chronological evolution of the domain and to the mathematical minimalism, and we check whether we can compute using only communication, that is, only symport and antiport rules. This leads to considering one of the most interesting classes of P systems, which we formally introduce here.

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

[2] By $|u|$ we denote the length of the string $u \in V^*$ for any alphabet V.

$$\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where:

1. O is the alphabet of objects,
2. μ is the membrane structure (of degree $m \geq 1$, with the membranes labeled $1, 2, \ldots, m$ in a one-to-one manner),
3. $w_1, \ldots, w_m$ are strings over O representing the multisets of objects present in the m compartments of μ in the initial configuration of the system,
4. $E \subseteq O$ is the set of objects supposed to appear in the environment in arbitrarily many copies,
5. $R_1, \ldots, R_m$ are the (finite) sets of rules associated with the m membranes of μ,
6. $i_o \in H$ is the label of a membrane of μ, which indicates the *output* region of the system.

The rules from R can be of two types, symport rules and antiport rules, of the forms specified above.

The rules are used in the nondeterministic maximally parallel manner. We define transitions, computations, and halting computations in the usual way. The number (or the vector of multiplicities) of objects present in region i_o in the halting configuration is said to be computed by the system by means of that computation; the set of all numbers (or vectors of numbers) computed in this way by Π is denoted by $N(\Pi)$ (by $Ps(\Pi)$, respectively).

We note here a new component of the system, the set E of objects which are present in the environment in arbitrarily many copies; because we move objects only across membranes and because we start with finite multisets of objects present in the system, we cannot increase the number of objects necessary for the computation if we do not provide a supply of objects, and this can be done by considering the set E. Because the environment is supposedly inexhaustible, the objects from E are inexhaustible; regardless of how many of them are brought into the system, arbitrarily many remain outside.

Another new feature is that this time the rules are associated with membranes, and not with regions, and this is related to the fact that each rule governs communication through a specific membrane.

The P systems with symport/antiport rules have a series of attractive characteristics: they are fully based on biological types of multiset processing rules; the environment plays a direct role in the evolution of the system; the computation is done only by communication, no object is changed, and the objects move only across membranes; no object is created or destroyed, and hence the conservation law is observed (as given in the previous sections, this is not valid for multiset rewriting rules because, for instance, rules of the form $a \to aa$ or $ff \to f$ are allowed).

11 An Example (Like a Proof. . .)

Because P systems with symport/antiport rules constitute an important class of P systems, it is worth considering an example; however, instead of a simple example, we directly give a general construction for simulating a *register machine.* In this way, we also introduce one of the widely used proof techniques for the universality results in this area. (Of course, the biologist can safely skip this section.)

Informally speaking, a register machine consists of a specified number of counters (also called registers) which can hold any natural number, and which are handled according to a program consisting of labeled instructions; the counters can be increased or decreased by 1 – the decreasing possible only if a counter holds a number greater than or equal to 1 (we say that it is nonempty) – and checked whether they are nonempty.

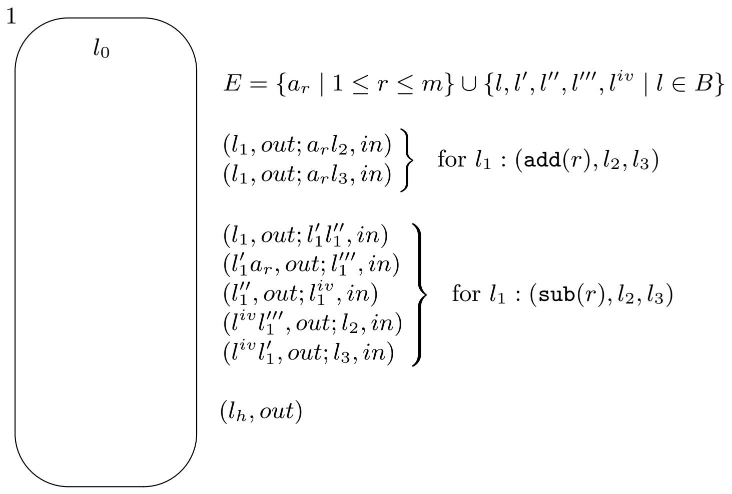
Formally, a (nondeterministic) *register machine* is a device $M = (m, B, l_0, l_h, R)$, where $m \geq 1$ is the number of counters, B is the (finite) set of instruction labels, l_0 is the initial label, l_h is the halting label, and R is the finite set of instructions labeled (hence uniquely identified) by elements from B. The labeled instructions are of the following forms:

– $l_1 : (\texttt{add}(r), l_2, l_3)$, $1 \leq r \leq m$ (add 1 to counter r and go nondeterministically to one of the instructions with labels l_2, l_3),
– $l_1 : (\texttt{sub}(r), l_2, l_3)$, $1 \leq r \leq m$ (if counter r is not empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3).

A counter machine generates a k-dimensional vector of natural numbers in the following manner: we distinguish k counters as output counters (without loss of generality, they can be the first k counters), and we start computing with all m counters empty, with the instruction labeled l_0; if the label l_h is reached, then the computation *halts* and the values of counters $1, 2, \ldots, k$ are the vector generated by the computation. The set of all vectors from $\mathbf{N}^k$ generated in this way by M is denoted by $Ps(M)$. If we want to generate only numbers (1-dimensional vectors), then we have the result of a computation in counter 1, and the set of numbers computed by M is denoted by $N(M)$. It is known (see [60]) that nondeterministic counter machines with $k + 2$ counters can compute any set of Turing computable k-dimensional vectors of natural numbers (hence machines with three counters generate exactly the family of Turing computable sets of numbers).

Now, a register machine can be easily simulated by a P system with symport/antiport rules. The idea is illustrated in Figure 4, where we have represented the initial configuration of the system, the rules associated with the unique membrane, and the set E of objects present in the environment.

The value of each register r is represented by the multiplicity of object $a_r, 1 \leq r \leq m$, in the unique membrane of the system. The labels from B,



$$E = \{a_r \mid 1 \leq r \leq m\} \cup \{l, l', l'', l''', l^{iv} \mid l \in B\}$$

$$\left.\begin{array}{l}(l_1, out; a_r l_2, in)\\(l_1, out; a_r l_3, in)\end{array}\right\}\ \text{for } l_1 : (\texttt{add}(r), l_2, l_3)$$

$$\left.\begin{array}{l}(l_1, out; l_1' l_1'', in)\\(l_1' a_r, out; l_1''', in)\\(l_1'', out; l_1^{iv}, in)\\(l^{iv} l_1''', out; l_2, in)\\(l^{iv} l_1', out; l_3, in)\end{array}\right\}\ \text{for } l_1 : (\texttt{sub}(r), l_2, l_3)$$

$$(l_h, out)$$

Fig. 4. An example of symport/antiport P system.

as well as their primed versions, are also objects of our system. We start with the unique object l_0 present in the system. In the presence of a label object l_1 we can simulate the corresponding instruction $l_1 : (\texttt{add}(r), l_2, l_3)$ or $l_1 : (\texttt{sub}(r), l_2, l_3)$.

The simulation of an add instruction is clear, so we discuss only a sub instruction. The object l_1 exits the system in exchange of the two objects $l_1' l_1''$ (rule $(l_1, out; l_1' l_1'', in)$). In the next step, if any copy of a_r is present in the system, then l_1' has to exit (rule $(l_1' a_r, out; l_1''', in)$), thus diminishing the number of copies of a_r by one, and bringing inside the object l_1'''; if no copy of a_r is present, which corresponds to the case when the register r is empty, then the object l_1' remains inside. Simultaneously, rule $(l_1'', out; l_1^{iv}, in)$ is used, bringing inside the "checker" l_1^{iv}. Depending on what this object finds in the system, either l_1''' or l_1', it introduces the label l_2 or l_3, respectively, which corresponds to the correct continuation of the computation of the register machine.

When the object l_h is introduced, it is expelled into the environment and the computation stops.

Clearly, the (halting) computations in Π directly correspond to (halting) computations in M; hence $Ps(M) = Ps(\Pi)$.

12 A Large Panoply of Possible Extensions

We have mentioned the flexibility and the versatility of the formalism of MC, and we have already mentioned several types of systems, making use of several types of rules, with the output of a computation defined in various ways.

We continue here in this direction, by presenting a series of possibilities of changing the form of rules and/or the way of using them. The motivation for such extensions comes both from biology, i.e., from the desire to capture more and more biological facts, and from mathematics and computer science, i.e., from the desire to have more powerful or more elegant models.

First, let us return to the basic target indications, *here, in,* and *out*, associated with the objects produced by rules of the form $u \to v$; *here* and *out* indicate precisely the region where the object is to be placed, but *in* introduces a degree of nondeterminism in the case where there are several inner membranes. This nondeterminism can be avoided by indicating also the label of the target membrane, that is, using target indications of the form in_j, where j is a label. An intermediate possibility, more specific than *in* but not completely unambiguous like in_j, is to assign the *electrical polarizations*, $+, -$, and 0, to both objects and membranes. The polarizations of membranes are given from the beginning (or can be changed during the computation), the polarization of objects is introduced by rules, using rules of the form $ab \to c^+c^-(d^0, tar)$. The charged objects have to go to any lower level membrane of opposite polarization, while objects with neutral polarization either remain in the same region or get out, depending on the target indication $tar \in \{here, out\}$ (this is the case with d in the previous rule).

A spectacular generalization, considered recently in [26], is to use indications in_j, for any membrane j from the system; hence the object is "teleported" immediately at any distance in the membrane structure. Also, commands of the form in^* and out^* were used, with the meaning that the object should be sent to (one of) the elementary membranes from the current membrane or to the skin region, respectively, no matter how far the target is.

We have considered the membrane dissolution action, represented by the symbol δ; we may imagine that such an action decreases the thickness of the membrane from the normal thickness, 1, to 0. A dual action can be also used, of increasing the thickness of a membrane, from 1 to 2. We indicate this action by τ. Assume that δ also decreases the thickness from 2 to 1, that the thickness cannot have other values than 0 (membrane dissolved), 1 (normal thickness), and 2 (membrane impermeable), and that when both δ and τ are introduced simultaneously in the same region, by different rules, their actions cancel, and the thickness of the membrane does not change. In this way, we can nicely control the work of the system: if a rule introduces a target indication *in* or *out* and the membrane which has to be crossed by the respective object has thickness 2, and hence is non-permeable, then the rule cannot be applied.

Let us look now to the catalysts. In the basic definition they never change their state or their place like ordinary objects do. A "democratic" decision is to also let the catalysts evolve within certain limits. Thus, *mobile catalysts* were proposed, moving across membranes like any object (but not themselves changing). The catalysts were then allowed to change their state, for instance, oscillating between c and $\bar{c}$. Such a catalyst is called *bistable*, and the natural generalization is to consider k-stable catalysts, allowed to change along k

given forms. Note that the number of catalysts is not changed; we do not produce or remove catalysts (provided they do not leave the system), and this is important in view of the fact that the catalysts are in general used for inhibiting the parallelism (a rule $ca \to cv$ can be used simultaneously at most as many times as copies of c are present).

There are several possibilities for controlling the use of rules, leading in general to a decrease in the degree of nondeterminism of a system. For instance, a mathematically and biologically motivated possibility is to consider a *priority* relation on the set of rules from a given region, in the form of a partial order relation on the set of rules from that region. This corresponds to the fact that certain reactions/reactants are more active than others, and can be interpreted in two ways: as a competition for reactants/objects, or in a strong sense. In the latter sense, if a rule r_1 has priority over a rule r_2 and r_1 can be applied, then r_2 cannot be applied, regardless of whether rule r_1 leaves objects which it cannot use. For instance, if $r_1 : ff \to f$ and $r_2 : cf \to cd\delta$, as in the example from Section 8, and the current multiset is $fffc$, because rule r_1 can be used, consuming two copies of f, we do not also use the second rule for the remaining fc. In the weak interpretation of the priority, the use of the second rule is allowed: the rule with the maximal priority takes as many objects as possible, and, if there are objects still remaining, the next rule in the decreasing order of priority is used for as many objects as possible, and we continue in this way until no further rule can be added to the multiset of applicable rules.

Also coming directly from bio-chemistry are the rules with *promoters* and *inhibitors*, written in the form $u \to v|_z$ and $u \to v|_{\neg z}$, respectively, where u, v, and z are multisets of objects; in the case of promoters, the rule $u \to v$ can be used in a given region only if all objects from z are present in the same region, and they are different from the (copies of) objects from u; in the inhibitors case, no object from z should be present in the region and be different from the objects from u. The promoting objects can evolve at the same time by other rules, or by the same rule $u \to v$ but by another instance of it (e.g., $a \to b|_a$ can be used twice in a region containing two copies of a, with each instance of $a \to b|_a$ acting on one copy of a and promoted by the other copy, but it cannot be used in a region where a appears only once).

An interesting combination of rewriting-communication rules are those considered in [77], where rules of the following three forms are proposed: $a \to (a, tar)$, $ab \to (a, tar_1)(b, tar_2)$, and $ab \to (a, tar_1)(b, tar_2)(c, come)$, where a, b, and c are objects, and tar, tar_1, and tar_2 are target indications of the forms *here, in,* and *out*, or in_j, where j is the label of a membrane. Such a rule just moves objects from one region to another, with rules of the third type usable only in the skin region; $(c, come)$ means that a copy of c is brought into the system from the environment. Clearly, these rules are different from the symport/antiport rules; for instance, the two objects ab from a rule $ab \to (a, tar_1)(b, tar_2)$ start from the same region, and can go in different directions, one up and the other down, in the membrane structure.

We are left with one of the most general type of rules, introduced in [11] under the name *boundary rules*, directly capturing the idea that many reactions take place on the inner membranes of a cell, depending maybe on the contents of both the inner and the outer regions adjacent to that membrane. These rules are of the form $xu[_i vy \rightarrow xu'[_i v'y$, where x, u, u', v, v', and y are multisets of objects and i is the label of a membrane. Their meaning is that in the presence of the objects from x outside and from y inside the membrane i, the multiset u from outside changes to u', and, simultaneously, the multiset v from inside changes to v'. The generality of this kind of rules is apparent, and it can be decreased by imposing various restrictions on the multisets involved.

There also are other variants considered in the literature, especially in the way of controlling the use of the rules, but we do not continue here in that direction.

13 P Systems with Active Membranes

We pass now to presenting a class of P systems, which, together with the basic transition systems and the symport/antiport systems, is one of the three central types of cell-like P systems considered in membrane computing. As in the above case of boundary rules, they start from the observation that membranes play an important role in the reactions which take place in a cell, and, moreover, they can evolve themselves, either by changing their characteristics or by dividing.

This last idea especially has motivated the class of *P systems with active membranes*, which are constructs of the form

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R),$$

where:

1. $m \geq 1$ (the initial degree of the system);
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure*, consisting of m membranes initially having neutral polarizations labeled (not necessarily in a one-to-one manner) with elements of H;
5. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *developmental rules*, of the following forms:
   (a) $[_h a \rightarrow v]_h^e$, for $h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*$
   (object evolution rules, associated with membranes and depending on the label and the charge of the membranes but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(b) $a[_h\ ]_h^{e_1} \rightarrow [_h b]_h^{e_2}$, for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
(*in* communication rules; an object is introduced in the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(c) $[_h a\ ]_h^{e_1} \rightarrow [_h\ ]_h^{e_2} b$, for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
(*out* communication rules; an object is sent out of the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[_h a\ ]_h^{e} \rightarrow b$, for $h \in H, e \in \{+, -, 0\}, a, b \in O$
(dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[_h a\ ]_h^{e_1} \rightarrow [_h b\ ]_h^{e_2} [_h c\ ]_h^{e_3}$, for $h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, and possibly of different polarizations; the object specified in the rule is replaced in the two new membranes possibly by new objects; the remaining objects are duplicated and may evolve in the same step by rules of type (a)).

The objects evolve in the maximally parallel manner, used by rules of type (a) or by rules of the other types, and the same is true at the level of membranes, which evolve by rules of types (b)–(e). Inside each membrane, the rules of type (a) are applied in parallel, with each copy of an object used by only one rule of any type from (a) to (e). Each membrane can be involved in only one rule of types (b)–(e) (the rules of type (a) are not considered to involve the membrane where they are applied). Thus, in total, the rules are used in the usual nondeterministic maximally parallel manner, in a bottom-up way (we use first the rules of type (a), and then the rules of other types; in this way, in the case of dividing membranes, the result of using first the rules of type (a) is duplicated in the newly obtained membranes). Also, as usual, only halting computations give a result, in the form of the number (or the vector) of objects expelled into the environment during the computation.

The set H of labels has been specified because it is possible to allow the change of membrane labels. For instance, a division rule can be of the more general form

(e') $[_{h_1} a\ ]_{h_1}^{e_1} \rightarrow [_{h_2} b\ ]_{h_2}^{e_2} [_{h_3} c\ ]_{h_3}^{e_3}$,
for $h_1, h_2, h_3 \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$.

The change of labels can also be considered for rules of types (b) and (c). Also, we can consider the possibility of dividing membranes in more than two copies, or even of dividing non-elementary membranes (in such a case, all inner membranes are duplicated in the new copies of the membrane).

It is important to note that in the case of P systems with active membranes, the membrane structure evolves during the computation, not only by decreasing the number of membranes, due to dissolution operations (rules of type (d)), but also by increasing the number of membranes by division. This increase can be exponential in a linear number of steps: using a division rule successively n steps, due to the maximal parallelism, we get 2^n copies of the same membrane. This is one of the most investigated ways of obtaining an exponential working space in order to trade time for space and solve computationally hard problems (typically **NP**-complete problems) in feasible time (typically polynomial or even linear time).

Some details can be found in Section 20, but we illustrate here the way of using membrane division in such a framework with an example dealing with the generation of all 2^n truth assignments possible for n propositional variables.

Assume that we have the variables $x_1, x_2, \dots, x_n$; we construct the following system (of degree 2):

$$\begin{aligned}
\Pi &= (O, H, \mu, w_1, w_2, R),\\
O &= \{a_i, c_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{\texttt{check}\},\\
H &= \{1, 2\},\\
\mu &= [_1 [_2\ ]_2]_1,\\
w_1 &= \lambda,\\
w_2 &= a_1 a_2 \dots a_n c_1,\\
R &= \{[_2 a_i]_2^0 \to [_2 t_i]_2^0 [_2 f_i]_2^0 \mid 1 \leq i \leq n\}\\
&\quad \cup \{[_2 c_i \to c_{i+1}]_2^0 \mid 1 \leq i \leq n-1\}\\
&\quad \cup \{[_2 c_n \to \texttt{check}]_2^0,\ [_2\texttt{check}]_2^0 \to \texttt{check}[_2\ ]_2^+\}.
\end{aligned}$$

We start with the objects $a_1, \dots, a_n$ in the inner membrane and we divide this membrane repeatedly by means of the rules $[_2 a_i]_2^0 \to [_2 t_i]_2^0 [_2 f_i]_2^0$; note that the object a_i used in each step is nondeterministically chosen, but each division replaces that object by t_i (for *true*) in one membrane and with f_i (for *false*) in the other membrane; hence after n steps the configuration obtained is the same regardless of the order of expanding the objects. Specifically, we get 2^n membranes with label 2, each one containing a truth assignment for the n variables. Simultaneously with the division, we have to use the rules of type (a) which update the "counter" c; hence at each step we increase by one the subscript of c. Therefore, when all variables have been expanded, we get the object `check` in all membranes (the rule of type (a) is used first, and after that the result is duplicated in the newly obtained membranes). In step $n+1$, this object exits each copy of membrane 2, changing its polarization to positive; this is meant to signal the fact that the generation of all truth assignments is completed, and we can start checking the truth values of (the clauses of) the propositional formula.

The previous example was chosen also for showing that the polarizations of membranes are not used while generating the truth assignments, though they might be useful after that; till now, this is the case in all polynomial time solutions to **NP**-complete problems obtained in this framework, in particular for solving SAT (satisfiability of propositional formulas in the conjunctive normal form). An important *open problem* in this area is whether or not the polarizations can be avoided. This can be done if other ingredients are considered, such as label changing or division of non-elementary membranes, but without adding such features the best result obtained so far is that from [3] where it is proved that the number of polarizations can be reduced to two.

14 A Panoply of Possibilities for Having a Dynamical Membrane Structure

Membrane dissolving and dividing are only two of the many possibilities of handling the membrane structures. One additional possibility investigated early is membrane *creation*, based on rules of the form $a \to [_h v]_h$, where a is an object, v is a multiset of objects, and h is a label from a given set of labels. Using such a rule in a membrane j, we create a new membrane, with label h, having inside the objects specified by v. Because we know the label of the new membrane, we know the rules which can be used in its region (a "dictionary" of possible membranes is given, specifying the rules to be used in any membrane with labels in a given set). Because rules for handling membranes are of a more general interest (e.g., for applications), we illustrate them in Figure 5, where the reversibility of certain pairs of operations is also made visible.

For instance, converse to membrane division, the operation of *merging* the contents of two membranes can be considered; formally, we can write such a rule in the form $[_{h_1} a]_{h_1} [_{h_2} b]_{h_2} \to [_{h_3} c]_{h_3}$, where a, b, and c are objects and h_1, h_2, and h_3 are labels (we have considered the general case, where the labels can be changed).

Actually, the merging operation can also be considered as the reverse of the *separation* operation, formalized as follows: let $K \subseteq O$ be a set of objects; a separation with respect to K is done by a rule of the form $[_{h_1}\ ]_{h_1} \to [_{h_2} K]_{h_2} [_{h_3} \neg K]_{h_3}$, with the meaning that the contents of membrane h_1 is split into two membranes, with labels h_2 and h_3, the first one containing all objects in K and the second one containing all objects not in K.

The operations of *endocytosis* and *exocytosis* (we use these general names, although in biology there are distinctions depending on the size of the objects and the number of objects moved; phagocytosis, pinocytosis, etc.) are also simple to formalize. For instance, $[_{h_1} a]_{h_1} [_{h_2}\ ]_{h_2} \to [_{h_2} [_{h_1} b]_{h_1}]_{h_2}$, for $h_1, h_2 \in H, a, b \in V$, is an endocytosis rule, stating that an elementary membrane labeled h_1 enters the adjacent membrane labeled h_2 under the control of object a; the labels h_1 and h_2 remain unchanged during this process; however, the object a may be modified to b. Similarly, the rule $[_{h_2} [_{h_1} a]_{h_1}]_{h_2} \to [_{h_1} b]_{h_1} [_{h_2}\ ]_{h_2}$, for $h_1, h_2 \in H, a, b \in V$, indicates an exocytosis operation: an elementary membrane labeled h_1 is sent out of a membrane labeled h_2 under the control of object a; the labels of the two membranes remain unchanged, but the object a from membrane h_1 may be modified during this operation.

Finally, let us mention the operation of *gemmation*, by which a membrane is created inside a membrane h_1 and sent to a membrane h_2; the moving membrane is dissolved inside the target membrane h_2, thus releasing its contents there. In this way, multisets of objects can be transported from a membrane to another one in a protected way: the enclosed objects cannot be processed by the rules of the regions through which the travelling membrane passes. The travelling membrane is created with a label of the form $@_{h_2}$, which indicates that it is a temporary membrane, having to get dissolved inside the membrane with label h_2. Corresponding to the situation from biology, in [13], [14] one considers only the case where the membranes h_1, h_2 are adjacent and placed directly in the skin membrane, but the operation can be generalized.

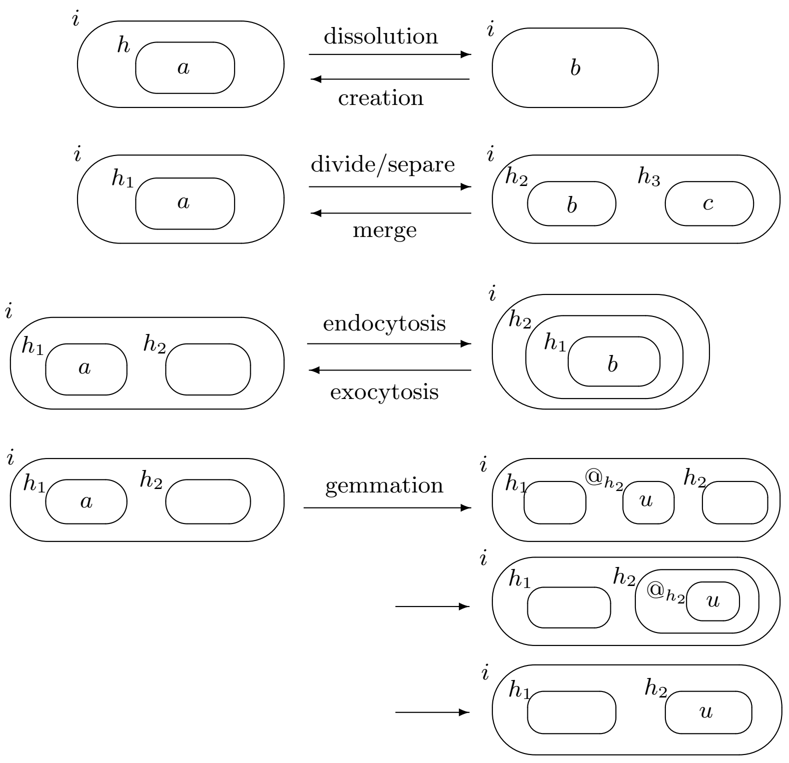


Fig. 5. Membrane handling operations.

A gemmation rule is of the form $a \to [_{@_{h_2}} u]_{@_{h_2}}$, where a is an object and u a multiset of objects (but it can be generalized by creating several travelling membranes at the same time, with different destinations); the result of applying such a rule is as illustrated in the bottom of Figure 5. Note that the crossing of one membrane takes one time unit (it is supposed that the travelling membrane finds the shortest path from the region where it is created to the target region).

Several other operations with membranes were considered, e.g., in the context of applications to linguistics, as well as in [47] and in other papers, but we do not enter into further details here.

15 Structuring the Objects

In the previous classes of P systems, the objects were considered atomic, identified only by their name, but in a cell many chemicals are complex molecules (e.g., proteins, DNA molecules, other large macromolecules), whose structure can be described by strings or more complex data, such as trees, arrays, etc. Also, from a mathematical point of view it is natural to consider P systems with string objects.

Such a system has the form

$$\Pi = (V, T, \mu, M_1, \dots, M_m, R_1, \dots, R_m),$$

where V is the alphabet of the system, $T \subseteq V$ is the terminal alphabet, μ is the membrane structure (of degree $m \geq 1$), $M_1, \dots, M_m$ are finite sets of strings present in the m regions of the membrane structure, and $R_1, \dots, R_m$ are finite sets of string-processing rules associated with the m regions of μ.

We have given here the system in general form, with a specified terminal alphabet (we say that the system is *extended*; if $V = T$, then the system is said to be *non-extended*), and without specifying the type of rules. These rules can be of various forms, but we consider here only two cases: rewriting and splicing.

In a *rewriting P system*, the string objects are processed by rules of the form $a \to u(tar)$, where $a \to u$ is a context-free rule over the alphabet V and tar is one of the target indications *here, in,* and *out.* When such a rule is applied to a string x_1ax_2 in a region i, we obtain the string x_1ux_2, which is placed in region i, in any inner region, or in the surrounding region, depending on whether tar is *here, in*, or *out*, respectively. The strings which leave the system do not come back; if they are composed only of symbols from T, then they are considered as generated by the system. The language of all strings generated in this way is denoted by $L(\Pi)$.

There are several differences from the previous classes of P systems: we work with *sets* of string objects, not with multisets; in order to introduce a string in the language $L(\Pi)$ we do not need to have a halting computation, because the strings do not change after leaving the system; each string is processed by only one rule (the rewriting is sequential at the level of strings), but in each step all strings from all regions which can be rewritten by local rules are rewritten by one rule.

In a *splicing P system*, we use splicing rules as those in DNA computing [38, 70], that is, of the form $u_1\#u_2\$u_3\#u_4$, where u_1, u_2, u_3, and u_4 are strings over V. For four strings $x, y, z, w \in V^*$ and a rule $r : u_1\#u_2\$u_3\#u_4$, we write

$$\begin{aligned}(x, y) \vdash_r (z, w) \text{ if and only if } & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\\ & z = x_1u_1u_4y_2,\ w = y_1u_3u_2x_2,\\ & \text{for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

We say that we splice x and y at the sites u_1u_2 and u_3u_4, respectively, and the result of the splicing (obtained by recombining the fragments obtained by cutting the strings as indicated by the sites) are the strings z and w.

In our case we add target indications to the two resulting strings, that is, we consider rules of the form $r : u_1\#u_2\$u_3\#u_4(tar_1, tar_2)$, with tar_1 and tar_2 one of *here, in,* and *out.* The meaning is as standard: after splicing the strings x, y from a given region, the resulting strings z, w are moved to the regions indicated by tar_1, tar_2, respectively. The language generated by such a system consists again of all strings over T sent into the environment during the computation, without considering only halting computations.

We do not give here an example of a rewriting or a splicing P system, but we move on to introducing an important extension of rewriting rules, namely, *rewriting with replication*, [49]. In such systems, the rules are of the form $a \to (u_1, tar_1)||(u_2, tar_2)|| \dots ||(u_n, tar_n)$, with the meaning that by rewriting a string x_1ax_2 we get n strings, $x_1u_1x_2, x_1u_2x_2, \dots, x_1u_nx_2$, which have to be moved in the regions indicated by targets $tar_1, tar_2, \dots, tar_n$, respectively. In this case we work again with halting computations, and the motivation is that if we do not impose the halting condition, then the strings $x_1u_ix_2$ evolve completely independently; hence we can replace the rule $a \to (u_1, tar_1)||(u_2, tar_2)|| \dots ||(u_n, tar_n)$ with n rules $a \to (u_i, tar_i), 1 \leq i \leq n$, without changing the language; that is, replication makes a difference only in the halting case.

The replicated rewriting is important because it provides the possibility to replicate strings, thus enlarging the workspace, and indeed this is one of the frequently used ways to generate an exponential workspace in linear time, used then for solving computationally hard problems in polynomial time.

Besides these types of rules for string processing, other kinds of rules were also used, such as insertion and deletion, context adjoining in the sense of Marcus contextual grammars [63], splitting, conditional concatenation, and so on, sometimes with motivations from biology, where several similar operations can be found, e.g., at the genome level.

16 Tissue-Like P Systems

We pass now to consider a very important generalization of the membrane structure, passing from the cell-like structure, described by a tree, to a tissue-like structure, with the membranes placed in the nodes of an arbitrary graph (which corresponds to the complex communication networks established among adjacent cells by making their protein channels cooperate, moving molecules directly from one cell to another, [53]). Actually, in the basic variant of tissue-like P systems, this graph is virtually a total one; what matters is the communication graph, dynamically defined during computations. In short, several (elementary) membranes – also called cells – are freely placed in a common environment; they can communicate either with each other or with the environment by symport/antiport rules. Specifically, we consider antiport rules of the form $(i, x/y, j)$, where i, j are labels of cells or at most one is zero, identifying the environment, and x, y are multisets of objects. This means that the multiset x is moved from i to j at the same time as the multiset y is moved from j to i. If one of the multisets x, y is empty, then we have, in fact, a symport rule. Therefore, the communication among cells is done either directly, in one step, or indirectly, through the environment: one cell throws some objects out and other cells can take these objects in the next step or later. As in symport/antiport P systems, the environment contains a specified set of objects in arbitrarily many copies. A computation develops as standard, starting from the initial configuration and using the rules in the nondeterministic maximally parallel manner. When halting, we count the objects from a specified cell, and this is the result of the computation.

The graph plays a more important role in so-called *tissue-like P systems with channel-states*, [33], which are constructs of the form

$$\Pi = (O, T, K, w_1, \ldots, w_m, E, syn, (s_{(i,j)})_{(i,j)\in syn}, (R_{(i,j)})_{(i,j)\in syn}, i_o),$$

where O is the alphabet of *objects*, $T \subseteq O$ is the alphabet of *terminal* objects, K is the alphabet of *states* (not necessarily disjoint of O), $w_1, \ldots, w_m$ are strings over O representing the initial multisets of objects present in the cells of the system (it is assumed that we have m cells, labeled with $1, 2, \ldots, m$), $E \subseteq O$ is the set of objects present in arbitrarily many copies in the environment, $syn \subseteq \{(i, j) \mid i, j \in \{0, 1, 2, \ldots, m\}, i \neq j\}$ is the set of links among cells (we call them *synapses*; 0 indicates the environment) such that for $i, j \in \{0, 1, \ldots, m\}$ at most one of $(i, j), (j, i)$ is present in syn, $s_{(i,j)}$ is the *initial state* of the synapse $(i, j) \in syn$, $R_{(i,j)}$ is a finite set of rules of the form $(s, x/y, s')$, for some $s, s' \in K$ and $x, y \in O^*$, associated with the synapse $(i, j) \in syn$, and, finally, $i_o \in \{1, 2, \ldots, m\}$ is the *output* cell.

We note the restriction that there is at most one synapse among two given cells, and the synapse is given as an ordered pair (i, j) with which a state from K is associated. The fact that the pair is ordered does not restrict the communication between the two cells (or between a cell and the environment), because we work here in the general case of antiport rules, specifying simultaneous movements of objects in the two directions of a synapse.

A rule of the form $(s, x/y, s') \in R_{(i,j)}$ is interpreted as an antiport rule $(i, x/y, j)$ as above, acting only if the synapse (i, j) has the state s; the application of the rule means (1) moving the objects specified by x from cell i (from the environment, if $i = 0$) to cell j, at the same time with the move of the objects specified by y in the opposite direction, and (2) changing the state of the synapse from s to s'.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m cells; in each time unit, a rule is used on each synapse for which a rule can be used (if no rule is applicable for a synapse, then no object passes over it and its state remains unchanged). Therefore, the use of rules is sequential at the level of each synapse, but it is parallel at the level of the system: all synapses which can use a rule must do so (the system evolves synchronously). The computation is successful if and only if it halts and the result of a halting computation is the number of objects from T present in cell i_o in the halting configuration (the objects from $O - T$ are ignored when considering the result). The set of all numbers computed in this way by the system Π is denoted by $N(\Pi)$. Of course, we can compute vectors, by considering the multiplicity of objects from T present in cell i_o in the halting configuration.

A still more elaborated class of systems, called *population P systems*, were investigated in a series of papers by F. Bernardini and M. Gheorghe (see, e.g., [10]) with motivations related to the dynamics of cells in skin-like tissues, populations of bacteria, and colonies of ants. These systems are highly dynamical; not only the links between cells, corresponding to the channels from the previous model with states assigned to the channels, can change during the evolution of the system, but also the cells can change their names, can disappear (get dissolved), and can divide, thus producing new cells; these new cells inherit, in a well specified sense, links with the neighboring cells of the parent cell. The generality of this model makes it rather attractive for applications in areas such as those mentioned above, related to tissues, populations of bacteria, etc.

17 Neural-Like P Systems

The next step in enlarging the model of tissue-like P systems is to consider more complex cells, for instance, moving the states from the channels between cells to the cells themselves – while still preserving the network of synapses. This suggests the neural motivation of these attempts, aiming to capture something from the intricate structure of neural networks, of the way the neurons are linked and cooperate in the human brain.

We do not recall the formal definition of a neural-like P system, but we refer to [67] for details, and here we present only the general idea behind these systems.

We again use a population of cells (each one identified by its label) linked by a specified set of synapses. This time, each cell has at every moment a state from a given finite set of states, contents in the form of a multiset of objects from a given alphabet of objects, and a set of rules for processing these objects.

The rules are of the form $sw \to s'(x, here)(y, go)(z, out)$, where s, s' are states and w, x, y, z are multisets of objects; in state s, the cell consumes the multiset w and produces the multisets x, y, z; the objects from multiset x remain in the cell, those of multiset y have to be communicated to the cells toward which there are synapses starting in the current cell; a multiset z, with the indication *out*, is allowed to appear only in a special cell, designated as the output cell, and for this cell the use of the previous rule entails sending the objects of z to the environment.

The computation starts with all cells in specified initial states, with initially given contents, and proceeds by processing the multisets from all cells, simultaneously, according to the local rules, redistributing the obtained objects along synapses and sending a result into the environment through the output cell; a result is accepted only when the computation halts.

Because of the use of states, there are several possibilities for processing the multisets of objects from each cell. In the *minimal* mode, a rule is chosen and applied once to the current pair (state, multiset). In the *parallel* mode, a rule is chosen, e.g., $sw \to s'w'$, and used in the maximally parallel manner: the multiset w is identified in the cell contents in the maximal manner, and the rule is used for processing all these instances of w. Finally, in the *maximal* mode, we apply in the maximally parallel manner all rules of the form $sw \to s'w'$, that is, with the same states s and s' (note the difference with the parallel mode, where in each step we choose a rule and we use only that rule as many times as possible).

There are also three ways to move the objects between cells (of course, we only move objects produced by rules in multisets with the indication *go*). Assume that we have applied a rule $sw \to s'(x, here)(y, go)$ in a given cell i. In the *spread* mode, the objects from y are nondeteterministically distributed to all cells j such that (i, j) is a synapse of the system. In the *one* mode, all the objects from y are sent to one cell j, provided that the synapse (i, j) exists. Finally, we can also *replicate* the objects of y, and each object from y is sent to all cells j such that (i, j) is an available synapse.

Note that the states ensure a powerful way to control the work of the system, that the parallel and maximal modes are efficient ways to process the multisets, and that the replicative mode of distributing the objects provides the possibility of increasing exponentially the number of objects in linear time. All together, these features make the neural-like P system both very powerful and very efficient computing devices. However, this class of P systems still waits for a systematic investigation – maybe starting with questioning their very definition, and changing this definition in such a way as to capture more realistic brain-like features.

18 Other Ways of Using a P System; P Automata

In all previous sections we have considered the various types of P systems as *generative devices*: starting from an initial configuration, because of the non-determinism of using the rules, we can proceed along various computations, at the end of which we get a result; in total, all successful computations provide a set of numbers, of vectors or numbers, or a language (set of strings), depending on the way the result of a computation is defined. This grammar oriented approach is only one possibility, mathematically attractive and theoretically important, but not useful from a practical point of view when dealing with specific problems to be solved and specific functions to be computed. However, a P system can also be used for computing functions and for solving problems (in a standard algorithmic manner).

Actually, besides the generative approach, there are two other general (related) ways of using a P system: in the *accepting* mode and in the *transducer* mode. In both cases, an input is provided to the system in a way depending on the type of systems at hand. For instance, in a symbol object P system, besides the initial multisets present in the regions of the membrane structure, we can introduce a multiset w_0 in a specified region, adding the objects of w_0 to the objects present in that region. The computation proceeds, and if it halts, then we say that the input is accepted (or recognized). In the transducer mode, we have not only to halt, but also to collect an output from a specified output region, internal to the system or the environment.

Now, an important distinction appears between systems which behave deterministically (at each moment at most one transition is possible, that is, either the computation stops, or it continues in a unique mode) and those which work in a nondeterministic way. Such a distinction does not make much sense in the generative mode, especially if only halting computations provide a result at the end: such a system can generate only a single result. In the case of computing functions or solving problems (e.g., decidability problems), the determinism is obligatory.

Again a distinction is in order: actually, we are not interested in the way the system behaves, deterministically or nondeterministically, but in the uniqueness and the reliability of the result. If, for instance, we ask whether or not a propositional formula in conjunctive normal form is satisfiable or not, we do not care how the result is obtained, but we want to make sure that it is the right one. Whether or not the truth assignments were created as in the example from Section 13, expanding the variables in a random order, is not relevant; what is important is that after n steps we get the same configuration. This brings to the stage the important notion of *confluence*. A system is *strongly confluent* if, starting from the initial configuration and behaving in a way which we do not care, it after a while reaches a configuration from where the computation continues in a deterministic way. Because we are only interested in the result of computations (e.g., in the answer, yes or no, to a decidability problem), we can relax the previous condition, to a *weak con-*

fluence property: regardless of how the system works, it always halts and all halting computations provide the same result. These notions will be invoked when discussing the efficiency of P systems, as in Section 20.

Let us consider here in some detail the accepting mode of using a P system. Given, for instance, a transition P system Π, let us denote by $N_a(\Pi)$ the set of all numbers accepted by Π in the following sense: we introduce a^n, for a specified object a, into a specified region of Π, and we say that n is accepted if and only if there is a computation of Π, starting from this augmented initial configuration, which halts. In the case of systems taking objects from the environment, such as the symport/antiport or the communicative ones [77], we can consider that the system accepts/recognizes the sequence of objects taken from the environment during a halting computation (if several objects are brought into the system at the same time, then all their permutations are accepted as substrings of the accepted string). Similar strategies can be followed for all types of systems, tissue-like and neural-like included (but P automata were first introduced in the symport/antiport case in [30]; see also [32]).

The above set $N_a(\Pi)$ was defined in general, for nondeterministic systems, but, clearly, in the accepting mode the determinism can be imposed (the nondeterminism is moved to the environment, to the "user," which provides an input, unique, but nondeterministically chosen, from which the computation starts). Note that the example of a P system with symport/antiport rules from Section 11 works in the same manner for an accepting register machine (a number is introduced in the first register and is accepted if and only if the computation halts); in such a case, the `add` instructions can be deterministic, that is, with labels l_2, l_3 identical (one simply writes $l_1 : (\mathtt{add}(r), l_2)$, with the continuation unique), and for this case the P system itself is deterministic.

19 Universality

The initial goal of membrane computing was to define computability models inspired from the cell biology, and indeed a large part of the investigations in this area was devoted to producing computing devices and examining their computing power, in comparison with the standard models in computability theory, Turing machines and their restricted variants. As it turns out, most of the classes of P systems considered are equal in power to Turing machines. In a rigorous manner, we have to say that they are Turing complete (or computationally complete), but because the proofs are always constructive, starting the constructions used in these proofs from universal Turing machines or from equivalent devices, we obtain universal P systems (able to simulate any other P system of the given type after introducing a "code" of the particular system as an input in the universal one). That is why we speak about *universality* results and not about computational completeness.

All classes of systems considered above, whether cell-like, tissue-like, or neural-like, with symbol objects or string objects, working in the generative or the accepting modes, with certain combinations of features, are known to be universal. The cell turns out to be a very powerful "computer," both when standing alone and in tissues.

In general, for P systems working with symbol objects, these universality results are proved by simulating computing devices known to be universal, and which either work with numbers or do not essentially use the positional information from strings. This is true/possible for register machines, matrix grammars (in the binary normal form), programmed grammars, regularly controlled grammars, and graph-controlled grammars (but not for arbitrary Chomsky grammars and for Turing machines, which can be used only in the case of string objects). The example from Section 11 illustrates a universality proof for the case of P systems with symport/antiport rules (with rules of sufficiently large weight; see below stronger results from this point of view).

We do not enter here into details other than specifying some notations which are already standard in membrane computing and, after that, mentioning some universality results of particular interest.

As for notations, the family of sets $N(\Pi)$ of numbers (we hence use the symbol N) generated by P systems of a specified type (P), working with symbol objects (O), having at most m membranes, and using features/ingredients from a given *list* is denoted by $NOP_m($*list-of-features*$)$. If we compute sets of vectors, we write $PsOP_m(\ldots)$, with Ps coming from "Parikh set." When the systems work in the accepting mode, one writes $N_aOP_m(\ldots)$, and when string objects are used, one replaces N with L (from "languages") and O with S (from "strings"), thus obtaining families $LSP_m(\ldots)$. The case of tissue-like systems is indicated by adding the letter t before P, thus obtaining $NOtP_m(\ldots)$, while for neural-like systems one uses instead the letter n. When the number of membranes is not bounded, the subscript m is replaced by $*$, and this is a general convention, used also for other parameters.

Now, the list of features can be taken from an endless pool: cooperative rules are indicated by *coo*; catalytic rules are indicated by *cat*, noting that the number of catalysts matters, and hence we use cat_r in order to indicate that we use systems with at most r catalysts; bistable catalysts are indicated by $2cat$ ($2cat_r$, if at most r catalysts are used); mobile catalysts are indicated by $Mcat$. When using a priority relation, we write *pri*. For the actions δ, τ we write simply δ, τ. Membrane creation is represented by *mcre*; endocytosis and exocytosis operations are indicated by *endo, exo*, respectively. In the case of P systems with active membranes, one directly lists the types of rules used, from (a) to (e), as defined and denoted in Section 13.

For systems with string objects, one write *rew*, $repl_d$, and *spl* for indicating that one uses rewriting rules, replicated rewriting rules (with at most d copies of each string produced by replication), and splicing rules, respectively.

In the case of (cell-like or tissue-like) systems using symport/antiport rules, we have to specify the maximal weight of the used rules, and this is done by writing $sym_p, anti_q$, meaning that symport rules of weight at most p and antiport rules of weight at most q are allowed.

There are many other features, with notations of the same type (as mnemonic as possible), which we do not recall here. Sometimes, when it is important to show in the name of the discussed family that a specific feature fe is *not* allowed, one writes nFe – for instance, one writes $nPri$ for not using priorities (note the capitalization of the initial name of the feature), $n\delta$, etc.

Specific examples of families of numbers (we do not consider here sets of vectors or languages, although, as we have said above, a lot of universality results are known for all cases) appear in the few universality results which we recall below. In these results, NRE denotes the family of Turing computable sets of numbers (the notation comes from the fact that these numbers are the length sets of recursively enumerable languages, those generated by Chomsky type-0 grammars or by many types of regulated rewriting grammars and recognized by Turing machines). The family NRE is also the family of sets of numbers generated/recognized by register machines. When dealing with vectors of numbers, hence with the Parikh images of languages (or with the sets of vectors generated/recognized by register machines), we write $PsRE$.

Here are some universality results (for the proofs, see the papers mentioned):

1. $NRE = NOP_1(cat_2)$, [31].
2. $NRE = NOP_3(sym_1, anti_1) = NOP_3(sym_2, anti_0)$, [4].
3. $NRE = NOP_3((a), (b), (c))$, [54].
4. $NRE = NSP_3(repl_2)$, [50].

In all these results, the number of membranes sufficient for obtaining the universality is pretty small. Actually, in all cases when the universality holds (and the code of a particular system is introduced in a universal system in such a way that the membrane structure is not modified), the hierarchy on the number of membranes collapses, because a number of membranes as large as the degree of the universal system suffices.

Still, “the number of membranes matters,” as we read already in the title of [43]: there are (sub-universal) classes of P systems for which the number of membranes induces an infinite hierarchy of families of sets of numbers (see also [44]).

20 Solving Computationally Hard Problems in Polynomial Time

The computational power (the “competence”) is only one of the important questions to be dealt with when defining a new computing model. The other fundamental question concerns the computing *efficiency*, the resources used for solving problems. In general, the research in natural computing is especially concerned with this issue. Because P systems are parallel computing devices, it is expected that they can solve hard problems in an efficient manner, and this expectation is confirmed for systems provided with ways for producing an exponential workspace in linear time.

We have discussed above three basic ways to construct such an exponential space in cell-like P systems, namely, membrane division (the separation operation has the same effect, as do other operations which replicate partially or totally the contents of a membrane), membrane creation (combined with the creation of exponentially many objects), and string replication. Similar possibilities are offered by cell division in tissue-like systems and by object replication in neural-like systems. Also the possibility to use a pre-computed exponential workspace, unstructured and non-active (e.g., with the regions containing no object) was considered.

In all these cases polynomial or pseudo-polynomial solutions to **NP**-complete problems were obtained. The first problem addressed in this context was `SAT` [66] (the solution was improved in several respects in other subsequent papers), but similar solutions are reported in the literature for the Hamiltonian Path and the Node Covering problems, the problem of inverting one-way functions, the Subset-sum problem and the Knapsack problem (note that the last two are numerical problems, where the answer is not of the yes/no type as in decidability problems), and for several other problems. Details can be found in [67, 72], as well as in the Web page of the domain, [82].

Roughly speaking, the framework for dealing with complexity matters is that of *accepting P systems with input*: a family of P systems of a given type is constructed starting from a given problem, and an instance of the problem is introduced as an input in such systems; working in a deterministic mode (or a *confluent* mode: some nondeterminism is allowed, provided that the branching converges after a while to a unique configuration, or, in the case of weak confluence, all computations stop in a determined time and give the same result) in a given time one of the answers yes/no is obtained in the form of specific objects sent to the environment. The family of systems should be constructed in a uniform mode (starting from the size of problem instances) by a Turing machine working in polynomial time. A more relaxed framework is that where a *semi-uniform* construction is allowed, carried out in polynomial time by a Turing machine, but starting from the instance to be solved (the condition to have a polynomial time construction ensures the "honesty" of the construction: the solution to the problem cannot be found during the construction phase).

This direction of research is very active at the present moment. More and more problems are being considered, the membrane computing complexity classes are being refined, characterizations of the $\mathbf{P}\neq\mathbf{NP}$ conjecture have been obtained in this framework, and improvements are being looked for. An important recent result concerns the fact that **PSPACE** was shown to be included in $\mathbf{PMC}_D$, the family of problems which can be solved in polynomial time by P systems with the possibility of dividing both elementary and non-elementary membranes. The **PSPACE**-complete problem used in this proof was `QSAT` (see [77, 5] for details).

There also are many *open problems* in this area. We have mentioned already the intriguing question about whether polynomial solutions to **NP**-complete problems can be obtained through P systems with active membranes without polarizations (and without label changing possibilities of other additional features). In general, the borderline between *efficiency* (the possibility to solve **NP**-complete problems in polynomial time) and *non-efficiency* is a challenging topic. Anyway, we know that membrane division cannot be avoided ("Milano theorem": a P system without membrane division can be simulated by a Turing machine with a polynomial slowdown; see [80, 81]).

21 Focusing on the Evolution

Computational power is of interest to theoretical computer science, and computational efficiency is of interest to practical computer science, but neither is of a direct interest to biology. Actually, this last statement is not correct: if a biologist is interested in simulating a cell – and this seems to be a major concern of biology today (see [48, 42] and other sources) – then the generality of the model (its comparison with the Turing machine and its restrictions) is directly linked to the possibility of algorithmically solving questions about the model. An example: is a given configuration reachable from the initial configuration? Imagine that the initial configuration represents a healthy cell and we are interested in knowing whether a sickness state is ever reached. Then, if both healthy and non-healthy configurations can be reached, the question arises whether we can find the "bifurcation configurations," and this is again a reachability issue. The relevance of such a "purely theoretical" problem is clear, and its answer depends directly on the generality (hence the power) of the model. Then, of course, the time needed for answering the question is a matter of computational complexity. So, both the power and the efficiency are, indirectly, of interest also to biologists, so we (the biologists, too) should be more careful when asserting that a given type of "theoretical" investigation is not of interest to biology.

Still, the immediate concern of biological research is the evolution of biological systems, their life, whatever this means, and not the result of a specific evolution. Alternatively stated, halting computations are of interest to computer science, whereas of direct interest to biology is the computation/evolution itself. Although membrane computing was not intended initially to deal with such issues, a series of recent investigations indicate a strong tendency toward considering P systems as dynamical systems. This does not concern only the fact that, besides the rules for object evolution, a more complete panoply of possibilities was imagined for making the membrane structure also evolve, with specific developments in the case of tissue-like and population P systems, where also the links between cells are evolving; but this concerns especially the formulation of questions which are typical for dynamical systems study. Trajectories, periodicity and pseudo-periodicity, stability, attractors, basins, oscillations, and many other concepts were brought in the framework of membrane computing – and the enterprise is not trivial, as these concepts were initially introduced in areas handled by means of continuous mathematics tools (mainly differential equations). A real program of defining discrete dynamical systems, with direct application to the dynamics of P systems, was started by V. Manca and his collaborators; we refer to Chapter 3 for details.

22 Recent Developments

Of course, the specification "recent" is risky, as it can soon become obsolete, but still we want to mention here some directions of research and some results which were not presented before – after repeating the fact that topics such as complexity classes and polynomial solutions to hard problems, dynamical systems approaches, and population P systems (in general, systems dealing with populations of cells, as in tissue-like or neural-like systems) are of a strong current interest which will probably lead to significant theoretical and practical results. To these trends we can add another general and yet not very structured topic: using non-crisp mathematics, handling uncertainty by means of probabilistic, fuzzy set, and rough set theories.

However, we want here to also point out a few more precise topics.

One of them concerns the role of time in P systems. The synchronization and the existence of a global clock are too strong assumptions (from a biological point of view). What about P systems where there exist no internal clocks and all rules have different times to be applied? This can mean both that the duration needed by a rule to be applied can differ from the duration of another rule and the extreme possibility that the duration is not known. In the first case, we can have a timing function assigning durations to rules; in the second case even such information is missing. How does the power of a system depend on the timing function? Are there time-free systems, which generate the same set of numbers regardless of what time function associates durations with its rules? Such questions are addressed in a series of papers by M. Cavaliere and D. Sburlan; see e.g., [22, 23].

Another powerful idea explored by M. Cavaliere and his collaborators is that of coupling a simple bio-inspired *system*, Sys, such as a P system without large computing power, with an *observer* Obs, a finite state machine which analyzes the configurations of the system Sys through the evolutions; from each configuration either a symbol or nothing (that is, the "result" of that configuration is the empty string λ) is produced; in a stronger variant, the observer can also reject the configuration and hence the system evolution, trashing it. The couple (Sys, Obs), for various simple systems and multiset processing finite automata, proved to be a very powerful computing device, universal even for very weak systems Sys. Details can be found in [19, 20].

An idea recently explored is that of trying to bound the number of objects used in a P system, and still computing all Turing computable numbers. The question can be seen as "orthogonal" to the usual questions concerning the number of membranes and the size of rules, since, intuitively, one of these parameters should be left free in order to codify and handle an arbitrary amount of information by using a limited number of objects. The first results of this type were given in [69] and they are surprising: in formal terms, we have $NRE = NOP_4(obj_3, sym_*, anti_*)$ (P systems with four membranes and symport and antiport rules of arbitrary weight are universal even when using only three objects). In turn, two objects (but without a bound on the number of membranes) are sufficient in order to generate all sets of vectors computed by so-called (see [36]) partially blind counter machines (for sets of numbers the result is not so interesting, because partially blind counter machines accept only semilinear sets of numbers, while the sets of vectors they accept can be non-semilinear).

Other interesting topics recently investigated which we only list here concern the reversibility of computations in P systems [52], energy accounting (associating quanta of energy to objects or to rules handled during the computation) [35, 34, 51], relations with grammar systems and with colonies [68], descriptional complexity, and non-discrete multisets [61, 27].

We close this section by mentioning the notion of *Sevilla carpet* introduced in [25], which proposes a way to describe the time-and-space complexity of a computation in a P system by considering the two-dimensional table of all rules used in each time unit of a computation. This corresponds to the Szilard language from language theory, with the complication now that we use several rules at the same step, and each rule is used several times. Considering all the information concerning the rules, we can get a global evaluation of the complexity of a computation, as illustrated, for instance, in [75] and [37].

23 Closing Remarks

The present chapter should be seen as a general overview of membrane computing, with the choice of topics intended to be as pertinent as possible, but, of course, not completely free of a subjective bias. The reader interested in further technical details, formal definitions, proofs, research topics and open problems, or details concerning the applications (and the software behind them) is advised to consult the relevant chapters of the book, as well as the comprehensive web page from `http://psystems.disco.unimib.it`. A complete bibliography of membrane computing can be found there, with many papers available for downloading (in particular, one can find there the proceedings volumes of the yearly Workshops on Membrane Computing, as well as of the yearly Brainstorming Weeks on Membrane Computing).

References

1. L.M. Adleman: Molecular Computation of Solutions to Combinatorial Problems. *Science*, 226 (November 1994), 1021–1024.
2. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.
3. A. Alhazov, R. Freund: On the Efficiency of P Systems with Active Membranes and Two Polarizations. In [59], 147–161.
4. A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: Communicative P Systems with Minimal Cooperation. In [59], 162–178.
5. A. Alhazov, C. Martín-Vide, L. Pan: Solving a PSPACE-Complete Problem by P Systems with Restricted Active Membranes. *Fundamenta Informaticae*, 58, 2 (2003), 67–77.
6. A. Alhazov, C. Martín-Vide, Gh. Păun, eds.: *Pre-Proceedings of Workshop on Membrane Computing*, WMC 2003, Tarragona, Spain, July 2003. Technical Report 28/03, Rovira i Virgili University, Tarragona, 2003.
7. I.I. Ardelean: The Relevance of Biomembranes for P Systems. *Fundamenta Informaticae*, 49, 1–3 (2002), 35–43.
8. J.-P. Banâtre, A. Coutant, D. Le Métayer: A Parallel Machine for Multiset Transformation and Its Programming Style. *Future Generation Computer Systems*, 4 (1988), 133–144.
9. J.-P. Banâtre, P. Fradet, D. Le Métayer: Gamma and the Chemical Reaction Model: Fifteen Years After. In [16], 17–44.
10. F. Bernardini, M. Gheorghe: Population P Systems. *Journal of Universal Computer Science*, 10, 5 (2004), 509–539.
11. F. Bernardini, V. Manca: Dynamical Aspects of P Systems. *BioSystems*, 70, 2 (2003), 85–93.
12. G. Berry, G. Boudol: The Chemical Abstract Machine. *Theoretical Computer Science*, 96 (1992), 217–248.
13. D. Besozzi: *Computational and Modeling Power of P Systems*. PhD Thesis, Univ. degli Studi di Milano, 2004.
14. D. Besozzi, C. Zandron, G. Mauri, N. Sabadini: P Systems with Gemmation of Mobile Membranes. In *Proc. ICTCS 2001*, Torino 2001 (A. Restivo, S.R. Della Rocca, L. Roversi, eds.), LNCS 2202, Springer, Berlin, 2001, 136–153.
15. C. Bonanno, V. Manca: Discrete Dynamics in Biological Models. *Romanian Journal of Information Science and Technology*, 5, 1-2 (2002), 45–67.
16. C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View. Lecture Notes in Computer Science*, 2235, Springer, Berlin, 2001.
17. L. Cardelli: Brane Calculus. In *Computational Methods in Systems Biology. International Conference CMSB 2004, Paris, France, May 2004, Revised Selected Papers*, LNCS 3082, Springer-Verlag, Berlin, 2005, 257–280.
18. M. Cavaliere: Evolution-Communication P Systems. In [71], 134–145.
19. M. Cavaliere, P. Leupold: Evolution and Observation – A New Way to Look at Membrane Systems. In [57], 70–87.
20. M. Cavaliere, P. Leupold: Evolution and Observation. A Non-Standard Way to Generate Formal Languages. *Theoretical Computer Science*, 321, 2-3 (2004), 233–248.

21. M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.: *Proceedings of the Brainstorming Week on Membrane Computing, Tarragona, February 2003.* Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003.
22. M. Cavaliere, D. Sburlan: Time-Independent P Systems. In [59], 239–258.
23. M. Cavaliere, D. Sburlan: Time and Synchronization in Membrane Systems. *Fundamenta Informaticae*, 64 (2005), 65–77.
24. R. Ceterchi, R. Gramatovici, N. Jonoska, K.G. Subramanian: Generating Picture Languages with P Systems. In [21], 85–100.
25. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla Carpets Associated with P Systems. In [21], 135–140.
26. L. Colson, N. Jonoska, M. Margenstern: λP Systems and Typed λ-Calculus. In [59], 1–18.
27. A. Cordón-Franco, F. Sancho-Caparrini: Approximating Non-Discrete P Systems. In [59], 288–296.
28. S. Crespi-Reghizzi, D. Mandrioli: Commutative Grammars. *Calcolo*, 13, 2 (1976), 173–189.
29. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, G. Vaszil: Cells in Environment: P Colonies. *Multiple Valued Logic and Soft Computing Journal*, to appear.
30. E. Csuhaj-Varju, G. Vaszil: P Automata or Purely Communicating Accepting P Systems. In [71], 219–233.
31. R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally Universal P Systems Without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
32. R. Freund, M. Oswald: A Short Note on Analysing P Systems. *Bulletin of the EATCS*, 78 (2003), 231–236.
33. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-Like P Systems with Channel-States. *Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University, 2004, 206–223, and *Theoretical Computer Science*, 330, 1 (2005), 101–116.
34. P. Frisco: *Theory of Molecular Computing. Splicing and Membrane Systems.* PhD Thesis, Leiden University, The Netherlands, 2004.
35. P. Frisco, S. Ji: Towards a Hierarchy of Info-Energy P Systems. In [71], 302–318.
36. S.A. Greibach: Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theoretical Computer Science*, 7 (1978), 311–324.
37. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: On Descriptive Complexity of P Systems. In [59], 321–331.
38. 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.
39. J. Hartmanis: About the Nature of Computer Science. *Bulletin of the EATCS*, 53 (1994), 170–190.
40. J. Hartmanis: On the Weight of Computation. *Bulletin of the EATCS*, 55 (1995), 136–138.
41. J. Hoffmeyer: Surfaces Inside Surfaces. On the Origin of Agency and Life. *Cybernetics and Human Knowing*, 5, 1 (1998), 33–42.
42. M. Holcombe: Computational Models of Cells and Tissues: Machines, Agents and Fungal Infection. *Briefings in Bioinformatics*, 2, 3 (2001), 271–278.
43. O.H. Ibarra: The Number of Membranes Matters. In [57], 218–231.

44. O.H. Ibarra: On Membrane Hierarchy in P Systems. *Theoretical Computer Science*, 334, 1-3 (2005), 115–129.
45. O.H. Ibarra: On Determinism Versus Nondeterminism in P Systems. *Theoretical Computer Science*, to appear. Available at `http://psystems.disco.unimib.it`.
46. O.H. Ibarra, H.-C.Yen, Z. Dang: The Power of Maximal Parallelism in P Systems. *Proceedings of the Eight Conference on Developments in Language Theory*, Auckland, New Zealand, 2004 (C.S. Calude, E. Calude, M.J. Dinneed, eds.), LNCS 3340, Springer, Berlin, 2004, 212–224.
47. M. Ionescu, T.-O. Ishdorj: Replicative-Distribution Rules in P Systems with Active Membranes. *Proc. of ICTAC2004, First Intern. Colloq. on Theoretical Aspects of Computing*, Guiyang, China, 2004.
48. H. Kitano: Computational Systems Biology. *Nature*, 420, 14 (2002), 206–210.
49. S.N. Krishna, R. Rama: P Systems with Replicated Rewriting. *Journal of Automata, Languages and Combinatorics*, 6, 3 (2001), 345–350.
50. S.N. Krishna, R. Rama, H. Ramesh: Further Results on Contextual and Rewriting P Systems. *Fundamenta Informaticae*, 64 (2005), 235–246.
51. A. Leporati, C. Zandron, G. Mauri. Simulating the Fredkin Gate with Energy-Based P systems. *Journal of Universal Computer Science*, 10, 5 (2004), 600–619.
52. A. Leporati, C. Zandron, G. Mauri: Universal Families of Reversible P Systems. *Proc. Conf. Universal Machines and Computations 2004*, Sankt Petersburg, 2004 (M. Margenstern, ed.), LNCS 3354, Springer, Berlin, 2005, 257–268.
53. W.R. Loewenstein: *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life.* Oxford University Press, New York, Oxford, 1999.
54. M. Madhu, K. Krithivasan: Improved Results About the Universality of P Systems. *Bulletin of the EATCS*, 76 (2002), 162–168.
55. V. Manca: String Rewriting and Metabolism. A Logical Perspective. In *Computing with Bio-Molecules. Theory and Experiments* (Gh. Păun, ed.), Springer, Singapore, 1998, 36–60.
56. S. Marcus: Bridging P Systems and Genomics: A Preliminary Approach. In [71], 371–376.
57. C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers. Lecture Notes in Computer Science*, 2933, Springer, Berlin, 2004.
58. C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón: Tissue P Systems. *Theoretical Computer Science*, 296, 2 (2003), 295–326.
59. G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revised Papers, Lecture Notes in Computer Science*, 3365, Springer, Berlin, 2005.
60. M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
61. T.Y. Nishida: Simulations of Photosynthesis by a K-subset Transforming System with Membranes. *Fundamenta Informaticae*, 49, 1-3 (2002), 249–259.
62. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, 20, 3 (2002), 295–306.
63. Gh. Păun: *Marcus Contextual Grammars.* Kluwer, Dordrecht, 1997.
64. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science–TUCS Report 208, November 1998, `www.tucs.fi`).

65. Gh. Păun: From Cells to Computers: Computing with Membranes (P Systems). *BioSystems*, 59, 3 (2001), 139–158.
66. Gh. Păun: P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.
67. Gh. Păun: *Computing with Membranes. An Introduction.* Springer, Berlin, 2002.
68. Gh. Păun: Grammar Systems vs. Membrane Computing: A Preliminary Approach. *Workshop on Grammar Systems*, MTA SZTAKI, Budapest, 2004, 225–245.
69. Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón: Symport/Antiport P Systems with Three Objects Are Universal. *Fundamenta Informaticae*, 64 (2005), 345–358.
70. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms.* Springer, Berlin, 1998.
71. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.: *Membrane Computing. International Workshop, WMC–CdeA 2002, Curtea de Argeş, Romania, Revised Papers. Lecture Notes in Computer Science*, 2597, Springer, Berlin, 2003.
72. M. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas.* Editorial Kronos, Sevilla, 2002.
73. B. Petreska, C. Teuscher: A Hardware Membrane System. In [57], 269–285.
74. A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro: BioAmbients – An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325 (2004), 141–167.
75. A. Riscos-Núñez: *Programacion celular. Resolucion eficiente de problemas numericos NP-complete.* PhD Thesis, Univ. Sevilla, 2004.
76. P. Sosik: The Computational Power of Cell Division in P Systems: Beating Down Parallel Computers? *Natural Computing*, 2, 3 (2003), 287–298.
77. P. Sosik, J. Matysek: Membrane Computing: When Communication Is Enough. In *Unconventional Models of Computation 2002* (C.S. Calude, M.J. Dinneen, F. Peper, eds.), LNCS 2509, Springer, Berlin, 2002, 264–275.
78. M. Tomita: Whole–Cell Simulation: A Grand Challenge of the 21st Century. *Trends in Biotechnology*, 19 (2001), 205–210.
79. G. Vaszil: On the Size of P Systems with Minimal Symport/Antiport. In *Pre-Proceedings of Workshop on Membrane Computing, WMC5, Milano, Italy*, June 2004, 422–431.
80. C. Zandron: *A Model for Molecular Computing: Membrane Systems.* PhD Thesis, Univ. degli Studi di Milano, 2001.
81. C. Zandron, C. Ferretti, G. Mauri: Solving NP-Complete Problems Using P Systems with Active Membranes. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 289–301.
82. The Web Page of Membrane Computing: `http://psystems.disco.unimib.it`.