Introduction to Membrane Computing

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: `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
E-mail: `gpaun@us.es`

Abstract. This is a comprehensive (and – supposed – friendly) introduction to membrane computing, meant to offer both to computer scientists and to non-computer scientists an up-dated overview of the domain. That is why the panoply of notions which are introduced here is rather large, but the presentation is informal, without any proof 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. Then, both some applications are briefly mentioned and a discussion is made about the attractiveness of this framework for (possible) applications, especially in biology.

1 (The Impossibility of) A Definition of Membrane Computing

Membrane computing 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, it deals with distributed and parallel computing models, processing mu ltisets 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 among compartment s (with the environment as well). 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 ca n be a hierarchical arrangement of membranes, like in a cell (hence described by a tree), or a net of membranes (placed in the nodes of a graph), like in a tissue, or in a neural net. The intuition behind the notion of a membr ane is that from biology, of a three–dimensional vesicle, but the concept itself is generalized/idealized to interpreting a membrane as a *separator* of two regions (of the Euclidean space), a finite "inside" an d an infinite "outside", also providing the possibility of a *selective communication* among 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 membrane computing contains a very large number of models. Thus, membrane computing is not a theory related to a specific model, it is a *framework* for devising compart mentalized models. Both because the domain is rather young (the trigger paper is [77], circulated first on web, but related ideas were considered before, in various contexts), but also as a genuine feature, bas ed both on the biological background and the mathematical formalism used, not only there are already proposed many types of P systems, but the flexibility and the versatility of P systems seem to be, in p rinciple, 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, with any good level of completeness.

However, there are a series of notions, notation, models which are already "standard", which have stabilized and can be considered as basic elements of membrane computing. This paper is devoted to presenting mainly such notions and models, together with the associated notation.

The presentation will be both historically and didactically organized, introducing first either notions which were investigated from the beginning in this area, or simpler notions, able to quickly offer an image of membrane computing to the reader who is not familiar with the domain.

The reader has surely noticed that all the previous discussion refers mainly to computer science (goals), and much less to biology. Membrane computing was not initiated as an area aiming to provide models to biology, in particular, models of the cell. Still in this moment, after a considerable development at the theoretical level, the domain is not fully prepared to offer such models to biology – but this is a strong tendency of the recent research and considerable advances towards such achievements were reported (the topic will be discussed later in more details).

2 Membrane Computing as Part of Natural Computing

Before entering into more specific elements of membrane computing, let us spend some time with the relationship of this area with, let us say, using the "local" terminology, the "outside". We have said above that membrane computing is part of computer science. However, the *genus proximus* is natural computing, the general attempt to learn computer science useful ideas, models, paradigms from the way nature – life especially – "computes", in various circumstances where substance and information processing can be interpreted as computations. Classic bio-inspired branches of natural computing are genetic algorithms (more generally, evolutionary computing, with well individualized sub-branches such as evolutionary programming) and neural computing. Both of them have a long history, which can be traced until unpublish ed works of Turing (see, e.g., [97]), many applications, and a huge bibliography. Both of them are a proof that "it is worth learning fr om biology", supporting the optimistic observation that many billions of year nature/life has adjusted certa in tools and processes which, correctly (luckily?)

abstracted and implemented in computer science terms, can p rove to be surprisingly useful in many applications.

A more recent branch of natural computing, with an enthusiastic beginning and unconfirmed yet computational applicability (we do not discuss here the by-products, such as the nanotechnology related developments), is DNA computing, whose birth certificate is related to Adleman experiment [1] of solving a (small) instance of the Hamiltonian path problem by handling DNA molecules in a laboratory. According to Hartmanis [50], [51], it was a *demo* that we can compute with bio-molecules, 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. However, at the theoretical level, DNA computing is beautifully developed (see, e.g., [85] and the proceedings of the yearly DNA Based Computers series of conferences). This is both due to the fact that DNA structure and processing suggest a series of new data structures (e.g., the double stranded sequence, with the pairs of corresponding symbols from the two strands being related through a complementarity relation) and operations (e.g., recombination, denaturation, hybridization), but also to the fact that the massive parallelism made possible by the efficiency of DNA as a support of information promises to be useful in solving computationally hard problems in a feasible time. Actually, at the theoretical level one can say that DNA computing started already in 1987, when T. Head [49] has proposed a language theoretic model of what he called the splicing operation (the recombination of DNA molecules, cut by restriction enzymes in fragments pasted together when the sticky ends match).

Both evolutionary computing and DNA computing are inspired from and related to handling DNA molecules. Neural computing considers the neurons are simple finite automata linked in networks of specific types. Thu s, 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", only interacting among them, 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 poin t where membrane computing enters the stage. Thus, membrane computing can be seen as an extension of DNA (m ore 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 as *alive*. It is very small and very intricate in its structure and functioning, has an elaborate internal activity and an exquisite interaction with the neighboring cells, and with the environment in general. It is fragile and robust at the same time, with a way to organize (control) the bio-chemical (and informational) processes which was polished during billions of years of evolution.

Any 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 "aggressed" during their journey by neighboring chemicals.

Then, the membranes allow a selective passage of substances among the compartmen ts delimited by them. This can be a simple selection by size, in the case of small molecules, or a much mor e intricate selection, through protein channels, which 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.

Much more: the membranes of a cell do not only delimit compartments where specif ic reactions take place in solution, hence *inside* the compartments, but many reactions in a cell develop *on the membranes*, catalyzed by the many proteins bound on 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 frequently en ough), and in order to create further "reaction surfaces". Anyway, biology contains many fascinating facts fr om a computer science point of view, and the reader is encouraged to check the validity of this assertion browsing, e.g., through [2], [64], [8].

Life means surfaces inside surfaces, as can be learned already from the ti tle of [52], while S. Marcus puts it in an equational form [69]: *Life = DNA software + membrane hardware.*

Then, there are cells living alone (unicellular organisms, such as ciliates, bac teria, etc.), but in general the cells are organized in tissues, organs, organisms, 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 org anization of the cell, all these suggest a lot of ideas, exciting from a mathematical point of view, and potentia lly useful from a computability point of view. Part of them were already explored in membrane computing, much mo re still wait for research efforts.

4 Some General Features of Membrane Computing Models

Still remaining at this general level, it is worth mentioning from the beginning some of the basic features of models investigated in this area, besides the essential use of membranes/compartmentalization.

We have mentioned above the notion of a multiset. The compartments of a cell contains substances (ions, small molecules, macromolecules) swimming in an aqueous solution; there is no ordering there, everything is close to everything, the concentration matters, 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 matter, 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 f orm of a string. For instance, if the objects a, b, c are present in, respectively, 5, 2, 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.

Both from the string representation of multisets and because of the bio-chemical background, where standard chemical reactions are common, the suggestion comes to process the multisets fro m 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 the fact that two copies of object a together with 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 pass to the multiset $a^4b^2c^8$. Similarly, by using the rule $bb \to aac$, we will 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 *non-determinism*. Which rules should be app lied 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 used both at the same time (for the mentioned multiset), as they compete for the "reactant" b. The standard solut ion to this problem in membrane computing is that *the rules and the objects are chosen in a non-determinist ic 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 (in a certain degree) non-deterministic, but it is also (in 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: in 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 only illustrate it with the previous multiset and pair of rules. Using these rules in the maximally parallel manner means to either use the first rule twice (thus involving four copies of a and both copies of b) or once the second rule (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 with 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. It deserves to be noted 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 also are other types of rules used in membrane computing (e.g., symport an d antiport rules), but we will discuss them later. Here we close with the observation that membrane computing d eals 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 reacti on-like) rules.

5 Computer Science Related Areas

Rewriting rules are standard rules for handling strings in formal language theory (although other types of rules are also used, both in formal lan guage theory and in P systems, such as insertion, deletion, context-adjoining, etc.). Also working with strings modulo the ordering of symbols is "old stuff": commutative languages (investigated, e.g., in [35]) are nothing else 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, formal power series.

Then, also the parallelism is considered in many areas of formal languages, and it is the main feature of Lindenmayer systems. These systems deserve a special discussion here, as they are a well developed branch of formal language theory inspired from biology, specifically, from the development of multi-cellular organisms (which can be described by strings of symbols). However, for L systems the cells are considered as symbols, not their structure is investigated but their organization in (mainly linear) patterns. 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 str ucture of the objects themselves, which leads to the category of P systems with string-objects.

However, a difference exists between the kind of parallelism in L systems and th at in P systems: there the parallelism is *total* – all symbols of a string should be processed at the same time, here we work with a maximal parallelism – we process as many objects as possible, but not necessari ly all of them.

Still closer to membrane computing are the multiset processing languages, the most known of them being Gamma [9, 10]. The standard rules of Gamma are of the form $u \rightarrow v(\pi)$, where u and v ar e 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 a great expressivity and, in a direct manner, computational universality . What Gamma does not have (at least in the initial versions) is distributivity. Then, membrane computing restricts the form of rules, on the one hand, as imposed by the biological roots, on the other hand , in search of mathematically simple and elegant models.

Also membranes appear, even in Gamma related models, and this is the case with CHAM, the Chemical Abstract Machine of Berry and Boudol, [16], the direct ancestor of membrane systems – wit h the mentioning that the membranes of CHAM ... are not membranes as in the cell biology, but they correspond to the contents of membranes, 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" co mes from "membrane") from [22] (see also [89] for a related approach), where process alge bra is used for investigating the processes taking place *on* membranes and *with* membranes of a cell .

The idea of devising a computing device based on compartmentalization through membranes was also suggested in [66].

Many related areas and many roots, with many common ideas and many differences. In some extent, membrane computing is a synthesis of part of these ideas, integrated in a framework directly inspired from the cell biology, paying the deserved attention to membranes (hence to distribution, hierarchization, communication, localization and to 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 a feasible time).

6 The Cell-Like Membrane Structure

We move now towards 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 wha t we can see when looking (through mathematical glasses, hence abstracting as much as necessary in order t o obtain a formal model) to a standard cell.

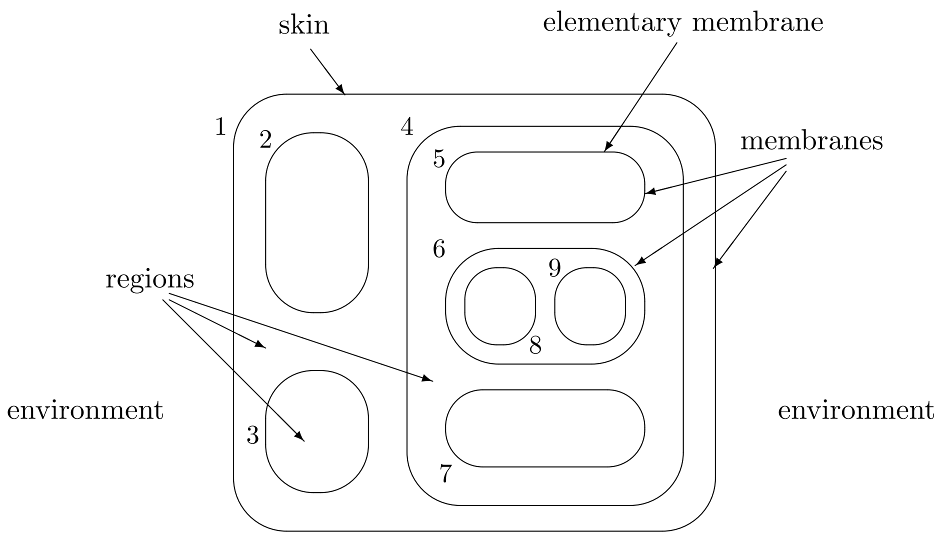


Figure 1: A membrane structure

Thus, as suggested by Figure 1, a membrane structure is a hierarchical ly 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 p laced inside the skin membrane (they correspond to the membranes present in a cell, around the nucleus, in Golg i apparatus, vesicles, mitochondria, etc.); a membrane without any other membrane inside it is said to be *elementary.* Each membrane determine s a compartment, also called *region*, the space delimited from above by it and from below by the membranes placed directly inside, if any exists. Clearly, the corre spondence membrane–region is one-to-one, that is why we sometimes use interchangeably the se terms.

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 stand ard labelling, but the labels can be more informative "names" associated with the membranes). Also, in the fi gure the labels are assigned in a one-to-one manner to membranes, but this is possible only in the case of membr ane structures which cannot grow (indefinitely), otherwise several membranes should have the same label (we will see later such cases). Due to the membrane–region correspondence, we identify by the same label a memb rane and its associated region.

Clearly, a hierarchical structure of membranes can be represented by a rooted tree; Figure 2 gives the tree which describes the membrane structure from Figure 1. The root of the tree is associated with the skin membrane and the leaves with the elementary membranes. In this way, graph–theoretic notions are brought into 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.

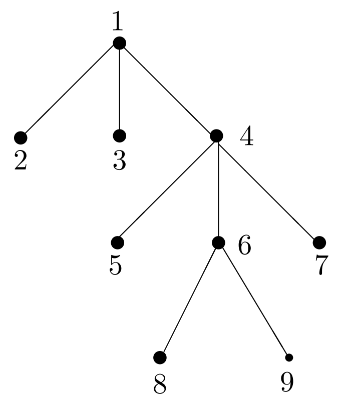


Figure 2: The tree describing the membrane structure from Figure 1

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

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

An important aspect should now be noted: the membranes from the same level can f loat around, that is, the tree representing the membrane structure is not oriented; in terms of parentheses exp ressions, two sub-expressions 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 an equivalent representation of the same membrane structure.

7 Evolution Rules and the Way of Using Them

In the basic variant of P systems, each region contains a multiset of symbol-obj ects, 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 by s ymbols 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 r ules: (1) multiset-rewriting rules (one uses to call 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 chemi cal reactions possible in the compartments of a cell, hence they are of the form $u \to v$, whe re u and v are multisets of objects. However, in order to make the compartments cooperate, we h ave to move objects across membranes, and to this aim we add *target indications* to the objects produced by a rule as above (to the objects from multiset v). These indications are: *here, i n, out*, with the meaning that an object having associated the indication *here* remains in the same r egion, one having associated the indication *in* goes immediately into a directly lower membra ne, non-deterministically chosen, and *out* indicates that the object has to exit the membrane, thus b ecoming an element of the region surrounding it. An example of evolution rule is $aab \rightarrow (a, here)(b, out)(c, here)(c, in)$ (this is the first of the rules considere d 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 b are con sumed (removed from the multiset of that region), and one copy of a, one of b, and two of c are pr oduced; 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, goin g to the surrounding region (indication *out*), and one of the new copies of c enters one of the child membranes, non-deterministically chosen. If no such child membrane exists, that is, the membrane with which the r ule is associated is elementary, then the indication *in* cannot be followed, and the rule cannot be applied. In t urn, if the rule is applied in the skin region, then b will exit into the environment of the system (and it is " lost" there, as it can never come back). In general, the indication *here* is not specified (an object wi thout 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 systems we are going to describe we do not provide similar rules for the environment, as we do not care about the objects present there; later we will consider types of P systems where also the environment takes part in the system evolution.

A rule as 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 acti on 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 d isappear at the same time with the membrane. The skin membrane is never dissolved.

The communication of objects through membranes reminds the fact that the 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 a 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 not quite realistic from a biological point of view, that is why there were also considered variants 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 (by symport/antiport rules).

It is also worth noting that evolution rules are stated in terms of *names o f objects*, while their application/execution is done using *copies of objects* – remem ber the example from Section 4, where the multiset $a^5b^2c^6$ was processed by a rule of the form $aab \rightarrow a(b, out)c(c, in)$, 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, non-deterministically choosing the rules and the objects.*

More specifically, this means that we assign objects to rules, non-deterministically choosing the objects and the rules, until no further assignment is possible. More 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 in order to apply the rules a number of times as indicated by their multiplicities, and (ii) the multiset is *maximal*, no further rule can be added to it (because of the lack of available objects).

Thus, an evolution step in a given region consists in finding a maximal applicab le multiset of rules, removing from the region all objects specified in the left hand of the chosen rules (with the 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 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 immediately upper membrane – provided that this membrane was not dissolved at t he same time, a case where we stop in the first upper membrane which was not dissolved (at least the skin r emains intact).

8 A Formal Definition of a Transition P System

Systems based on multiset-rewriting rules as above are usually called *trans ition 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 finit e non-empty alphabet of abstract symbols identifying the objects), the membrane structure (it can be represented in many ways, but the most used one is by a string of labelled matching parentheses), the multisets of objects prese nt in each region of the system (represented in the most compact way by strings of symbol-o bjects), the sets of evolution rules associated with each region, as well as the indication about the way the o utput is defined – see below.

Formally, a *transition P system* (of degree m) 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 non-empty) alphabet of *objects*,
2. $C \subset O$ is the set of *catalysts*,
3. μ is a membrane structure, consisting of m membranes, labelled with $1, 2, \ldots, m$; one says 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 o f objects* present in the regions $1, 2, \ldots, m$ of the membrane structure,
5. $R_1, R_2, \ldots, R_m$ are finite *sets of evolution rules* associated wi th the regions $1, 2, \ldots, m$ of the membrane structure,
6. i_o is either one of the labels $1, 2, \ldots, m$, and then the respective r egion is the *output region* of the system, or it is 0, and then the result of a computation is collected in the environment of the system.

The rules are of the form $u \rightarrow v$ or $u \rightarrow v\delta$, 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

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

$u \in O - C$), or catalytic (of the form $ca \rightarrow cv$ or $ca \rightarrow cv\delta$, with $a \in O - C, c \in C, v \in ((O - C) \times Tar)^*$) ; 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 th e label of an elementary membrane of μ).

In general, the membrane structure and the multisets of objects from its compart ments identify a *configuration* of a P system. The *initial configuration* is given by specifying the membra ne structure and the multisets of objects available in its compartments at the beginning of a computation, hence $(\mu, w_1, \ldots, w_m)$. During the evolution of the system, by means of applying the rules, both the multisets of o bjects 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 a pictorial way in Figure 3. It is important to note that adding to the initial configuration the set of rules, placed in the corresponding regions, we have a complete and co ncise presentation of the system (the indication of the output region can also be added in a suitable manner, for instance, writing "output" inside it).

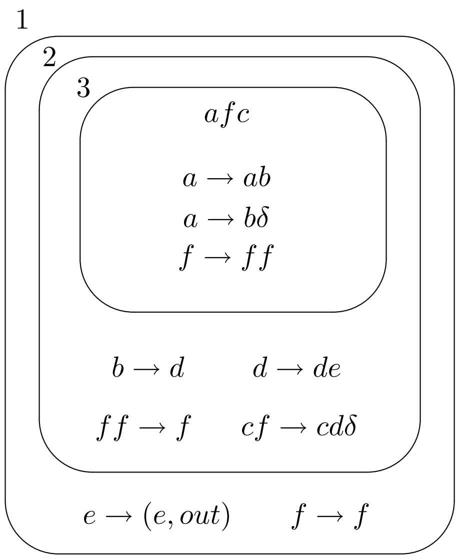


Figure 3: The initial configuration of a P system, rules included

9 Defining Computations and Results of Computations

In their basic variant, membrane systems are synchronous devices, in the sense t hat 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 ea ch region, in a *non-deterministic* and *maximally parallel manner*. As explained in th e previous sections, this means that the objects to evolve and the rules governing this evolution are chosen in a non-deterministic way, and this choice is "exhaustive" in the s ense that, after the choice was made, no rule can be applied in the same evolution st ep to the remaining objects.

A sequence of transitions constitutes a *computation.* A computation is successful if it halts, it reaches a configuration where no rule can be appli ed to the existing objects, and the output region i_o still exists in the halting configuration (in the case when i_o is the label of a membrane, it can be dissolved during the computation). With a success ful 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 d uring 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 numb ers. The objects which leave the system can also be arranged in a sequence according to the moments when they exit the skin membrane, and in this case the result is a string (if several objects exit at the same tim e, then all their permutations are accepted as a substring of the result). Note that non-halting computations provi de no output (we cannot know when a number is "completely computed" before halting), and if the output membrane is dissolved during the computation, then the computation aborts, no result is obtained (of course, this makes sense only in the case of the internal output).

A possible extension of the definition is to consider a *terminal* set of ob jects, $T \subseteq O$, and to count only the copies of objects from T, discarding the objects from $O-T$ pre sent 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 non-determinism of the application of rules, starting from an initial configuration, we can get several successful computations, hence several results. Thus, a P sys tem *computes* (one also uses to say *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 la nguage is important in view of the qualitative difference between the "loose" data structure we use inside the system (vectors of numbers) and th e 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 Pi in the above way. When we consider the vector of multiplicities of objects from the output region, we w rite $Ps(\Pi)$. In turn, in the case when we take as (external) output the strings of objects leaving the system , then 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 we can apply rules. Specifically, we can repeatedly apply the rule $a \rightarrow ab$ in parallel with $f \rightarrow ff$, and in this way the number of copies of b grows each step by o ne, while the number of copies of f is doubled in each step. If we do not apply the rule $a \rightarrow b\delta$ (again in parallel with $f \rightarrow ff$), which dissolves the membrane, then we can continue in this way forever. Thus, in order to ever h alt, we have to dissolve membrane 3. Assume that this happens after $n \geq 0$ steps of using the rules $a\ raab$ and $f \rightarrow ff$. When membrane 3 is dissolved, its contents ($n+1$ copies of b, 2^{n+1} copi es of f, and one copy of the catalyst c) are left free in membrane 2, which now can start using its rules. In the next step, all objects b become d. Let us examine the rule s $ff \rightarrow f$ and $cf \rightarrow cd\delta$. The second rule dissolves membrane 2, hence passes the contents of this membrane to membrane 1. If among the objects which arrive in membrane 1 there is at least one copy of

f, then the r ule $f \to f$ 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 $cf \to cd\delta$ s hould 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 fo r all pairs of f available, that is, in each step the number of copies of f is divided by 2. This is alrea dy done once in the step when all copies of b become d, and will be done from now on as long as at le ast two copies of f are present. Simultaneously, in each step each d produces one copy of e. This pr ocess can continue until we get a configuration with only one copy of f present; at that step we have t o use the rule $cf \to cd\delta$ (the rule $ff \to f$ is no longer applicable), hence also membrane 2 is dissolved . Because we have applied the rule $d \to de$, in parallel for all copies of d (there are $n+1$ such copi es), during $n+1$ steps, we have $(n+1)(n+1)$ copies of e, $n+2$ copies of d (one of them was produced by the rule $cf \to cd\delta$), and one copy of c present in the skin membrane of the system (the unique membrane still present). The objects e are sent out, and the computation halts. Therefo re, we compute in this way the number $(n+1)^2$, for some $n \geq 0$, 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 sub stances through membranes, and one of the most interesting ways to handle this trans-membrane communication is by coup ling molecules. The process by which two molecules pass together across a membrane (through a specific prote in channel) is called *symport*; when the two molecules pass simultaneously through a protein channel, but in opp osite 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 t he latter; 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), (x, out), and antiport rules of the form $(z, out; w, in)$, where x, z, w are multisets of arbitr ary size; one uses to say that $|x|$ is the *weight* of a symport rule as above, 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, out*: we consider multiset rewriting rules of the form $u \to v$ (or $u \to v\delta$) without target indications associated with the objects from v, as well as symport/antiport rules for communicating the ob jects among compartments. Such systems, called *evolution–communication* P systems, were considered in [23] (for various restricted types of rules of the two forms).

Here we do not go into this direction, but we stay closer both to the chronologi cal evolution of the domain, and to the mathematical minimalism, and we check whether we can compute using only communic ation, 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.

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

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

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

where:

1. O is the alphabet of objects,
2. μ is the membrane structure (of degree $m \geq 1$, with the membranes la belled in a one-to-one manner with $1, 2, \ldots, m$),
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 mem branes 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 as specified above.

The rules are used in the non-deterministic maximally parallel manner. In the usual way, we define transitions, computations, and halting computations.

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 along that computation; the set of all numbers (resp., vectors or num bers) computed in this way by Π is denoted by $N(\Pi)$ (resp., by $Ps(\Pi)$).

We remark here a new component of the system, the set E of objects which are present in the environment in arbitrarily many copies; because we only move objects 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 supposed inexhaustible, the objects from E are supposed inexhaustible, irrespective 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 the membrane s, not with the regions, and this is related to the fact that each rule governs the communication through a specific membrane.

The P systems with symport/antiport rules have a series of attractive characteri stics: they are fully based on biological types of multiset processing rules; the environment takes a direct pa rt into the evolution of the system; the computation is done only by communication, no object is changed, the objects only move across membranes; no object is created or destroyed, hence the conservation low is obse rved (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, but by using some dummy objects d available in arbitrarily many copies, we can take care of the conservation low by writing, e.g., $da \to aa$ and $ff \to fd$).

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 (actually, it is not very easy to find simple symport/antiport systems computing non-trivial sets of numbers), we give directly a general construction, for simulating a *register machine*. In this way, we also introduce one of the widely used proof techniques for the universality r esults in this area. (Of course, the biologist reader 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 ar e handled according to a program consisting of labelled instructions; the counters can be increased or decreased by 1 – the decreasing being possible only if a counter holds a number greater than or equal to 1 (we say that it is non-empty) –, and checked whether they are non-empty.

Formally, a (non-deterministic) *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 labelled (hence uniquel y identified) by elements from B (R is also called the *program* of the machine). The labelled instructions are of the following forms:

– $l_1 : (\mathtt{add}(r), l_2, l_3)$, $1 \leq r \leq m$ (add 1 to counter r and go non-deterministically to one of the instructions with labels l_2, l_3),

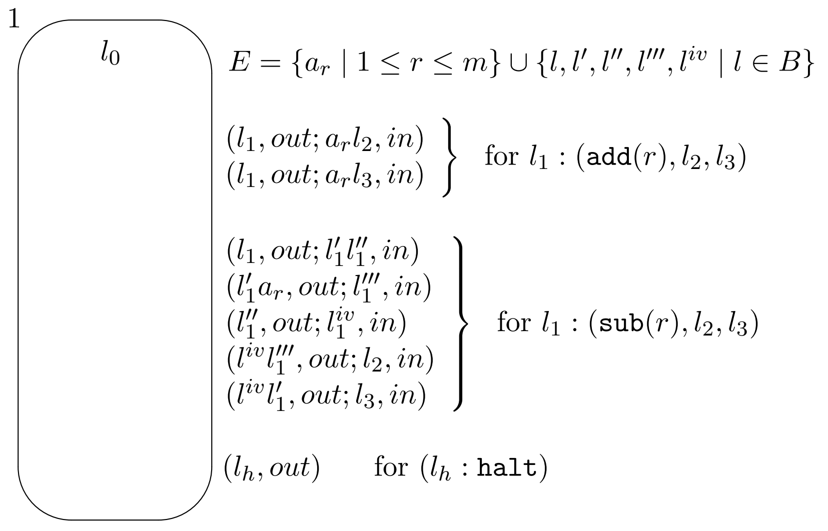
– $l_1 : (\mathtt{sub}(r), l_2, l_3)$, $1 \leq r \leq m$ (if counter r is not empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),

– $l_h : \mathtt{halt}$ (the halt instruction, which can only have the label l_h).

A counter machine generates a k-dimensional vector of natural numbers in the following manner: we distinguish k counters as output counters (withou t loss of generality, they can be the first k counters), and we start computing with all m counters being empty, with the instruction labelled by l_0; if the computation reaches the instruction $l_h : \mathtt{halt}$ (we say that it halts), then the values of counters $1, 2, \ldots, k$ is the vector generated by the computation. The set of al l 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 co mputed by M in this way is denoted by $N(M)$. It is known (see [73], [40]) that non-determi nistic counter machines with $k + 2$ counters can compute any set of Turing computable k-dimensional vectors of nat ural 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 configu ration of the system, the rules associated with the unique membrane, as well as 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, as well as primed version of them, are also objects of our system. We start with the unique object l_0 present in the system. In the



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

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

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

$$(l_h, out) \qquad \text{for } (l_h : \texttt{halt})$$

Figure 4: An example of symport/antiport P system

presence of a label–objec t 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 that 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'''; i f 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 halt instruction is reached, that is, the object l_h is introduced, t his object is just expelled into the environment and the computation stops.

Clearly, the (halting) computations in Π directly correspond to (halting) co mputations in M, hence $N(M) = N(\Pi)$.

12 A Large Panoply of Possible Extensions

We have mentioned the flexibility and the versatility of the formalism of membrane computing, 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 mo tivation for such extensions comes both from biology, from the natural desire to capture more and more biolog ical facts, and from mathematics and computer science, from the desire to have more powerful or more elegant models.

First, let us return to the basic target indications, *here, in, out*, assoc iated with the objects produced by rules of the form $u \to v$; *here* and *out* precisely indicate the r egion where the object is to be placed, but *in* introduces a degree of non-determinism in the case when the re are several inner membranes. This non-determinism can be avoided by indicating also the label of the target m embrane, that is, using target indications of the form in_j, where j is a label. An intermediate possibilit y, more specific than *in* but not completely unambiguous like in_j, is to assign both to objects and to membranes *electrical polarizations*, $+, -$, and 0. The pola rizations 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 $ab \rightarrow c^+c^-(d^0, tar)$. 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 previou s rule).

A spectacular generalization, considered recently in [33], is to use in dications in_j, for j being any membrane from the system, hence the object is "teleported" immediately at a ny distance in the membrane structure; also, commands of the form in^* and out^* were used, with the mea ning that the object should be sent to (one of) the elementary membranes from the current membrane or to the sk in region, respectively, no matter how far the target is.

Then, we have considered the membrane dissolution action, represented by the sym bol 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 indicat e this action by τ. Assume that δ also decreases the thickness from 2 to 1, that the thickness canno t have other values than 0 (membrane dissolved), 1 (normal thickness), and 2 (membrane impermeable), and that when bo th δ and τ are introduced simultaneously in the same region, by different rules, then their actions cancel , the thickness of the membrane does not change. In this way, we get a nice possibility to control the work of t he 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, hence it is non-permeable, then the rule cannot be applied.

Let us look now to the catalysts. In the basic definition they never change thei r state or their place, like ordinary objects do. A "democratic" decision is to let also the catalysts to evolve – in certain li mits. Thus, *mobile catalysts* were proposed, moving across membranes like any object (but still not changing t hemselves). Then, the catalysts were allowed to change their state, for instance, oscillating between c and $barc$. Such a catalyst is called *bi-stable*, and the natural generalization is to consider k-stable catalysts, allowed to change along k given forms. Note that in all cases the number of catalysts is not cha nged, we do not produce or remove catalysts (unless if they leave the system), and this is important in view of th e fact that the catalysts are in general used for inhibiting the parallelism (a rule $ca \rightarrow cv$ can be used si multaneously at most as many times as many copies of c are present).

There are several possibilities for controlling the use of rules, in general, le ading to a decrease of the degree of non-determinism of a system. For instance, a mathematically and biologically mot ivated 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 i n 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, irrespective whether rule r_1 leaves objects which it cannot use. For instance , if $r_1 : ff \rightarrow f$ and $r_2 : cf \rightarrow cd\delta$, like 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 remainin g fc. In the weak interpretation of the priority, this is allowed: the rule with the maximal priority takes as ma ny objects as possible, and, if still there are remaining objects, then the next rule in the decreasing order of priority is used for as many objects as possible, and, if still remain unused objects, we continue in this way until no further rule can be added to the multiset of applicable rules.

Also directly coming from bio-chemistry are the rules with *promoters* and *inhibitors*, written in the form $u \rightarrow v|_z$ and $u \rightarrow v|_{\neg z}$, respectively, where u, v, z are multisets of objects; in the case of promoters, the rule $u \rightarrow v$ 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 different from the objects from u. The promoting objects can evolve at the same time by other rules, or by the same rule $u \rightarrow v$, but by another instance of it (e.g., $a \rightarrow b|_a$ can be used twice in a region containing two copies of a, with each instance of $a \rightarrow b|_a$ acting on one copy of a and promoted by the other copy, but cannot be used at all in a region where a appears only once).

An interesting combination of rewriting–communication rules are those considere d in [92], where rules of the following three forms are proposed: $a \rightarrow (a, tar)$, $ab \rightarrow (a, tar_1)(b, tar_2)$, $ab \rightarrow (a, tar_1)(b, tar_2)(c, come)$, where a, b, c are objects, and tar, tar_1, tar_2 are target indications of the f orms *here, in, out* or even in_j, with j the label of a membrane. Such a rule just moves objects from a region to another one, with the mentioning that rules of the third type can be used only in the skin region and the indicat ion $(c, come)$ means that a copy of c is brought into the system from the environment. Clearly, these rules are differ ent from the symport/antiport rules; for instance, the two objects ab from a rule $ab \rightarrow (a, tar_1)(b, tar_2)$ start from the same region, and can go into different directions, one up and one down in the membrane structure.

We have left in the end one of the most general type of rules, introduced in [15] under the name of *boundary rules*, directly capturing the idea that many reactions take pl ace on the inner membranes of a cell, maybe depending on the contents of both the inner and the o uter region adjacent to that membrane. These rules are of the form $xu[_i vy \rightarrow xu'[_i v'y$, where x, u, u', v, v', y ar e multisets of objects and i is the label of a membrane. The meaning is that in the presence of the objects from x outside and of objects from y inside the membrane i, the multiset u from outside changes to multiset u' and, simultaneously, the multiset v from inside is changed into v'. The generality of this kind of rules is apparent – and it can be decreased by imposing various restrictions on the involved multisets.

There also are other variants considered in the literature, especially in what c oncerns the way of controlling the use the rules, but we do not continue here in this 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. Like in the above case of boundary rules, they start from the observation that the membranes play an important role in the reactions which take place in a cell, and, moreover, they can evolve themselves, either changing their characteristics or even getting divided.

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

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

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 having initially neutral polarizations, labelled (not necessarily in a one-to-one manner) with elements of H;

5. $w_1, \dots, w_m$ 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) $[_h a \to v]_h^e$,
   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) $a[_h\ ]_h^{e_1} \to [_h b]_h^{e_2}$,
   for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
   (*in* communication rules; an object is introduced in the membrane, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

   (c) $[_h a\ ]_h^{e_1} \to [_h]_h^{e_2} b$,
   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, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

   (d) $[_h a\ ]_h^e \to b$,
   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) $[_h a\ ]_h^{e_1} \to [_h b\ ]_h^{e_2} [_h c\ ]_h^{e_3}$,
   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, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly 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 the parallel way, with each copy of an object being used by only one rule of any type from (a) to (e). Each membrane can be involved in only one rule of types (b), (c), (d), (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 non-deterministic maximally parallel manner, in a bottom-up way (first we use the rules of type (a), and then the rules of other types; in this way, in the case of dividing membranes, in the newly obtained membranes we duplicate the result of using first the rules of type (a)). 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 also possible to allow th e change of membrane labels. For instance, a division rule can be of the more general form

(e') $[_{h_1} a\ ]^{e_1}_{h_1} \rightarrow [_{h_2} b\ ]^{e_2}_{h_2}[_{h_3} c\ ]^{e_3}_{h_3}$,
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-elementar y 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 d issolution operations (rules of type (d)), but also increasing the number of membranes, by division. This increase ca n be exponential in a linear number of steps: using successively a division rule, due to the maximal parallel ism, in n steps we get 2^n copies of the same membrane. This is one of the most investigated ways of obtain ing an exponential working space in order to trade time for space and solve computationally hard problems (typica lly **NP**-complete problems) in a feasible time (typically polynomial or even linear).

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

Assume that we have the variables $x_1, x_2, \ldots, x_n$; we construct the followin g system, of degree 2:

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

We start with the objects $a_1, \ldots, a_n$ in the inner membrane and we divide th is membrane, repeatedly, by means of rules $[_2 a_i]^0_2 \rightarrow [_2 t_i]^0_2[_2 f_i]^0_2$; note that the object a_i used i n each step is non-deterministically 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 obtained configura tion is the same irrespective which was the order of expanding the objects. Specifically, we get 2^n membran es with label 2, each one containing a truth–assignment for the n variables. Actually, simultaneously w ith the division, we have to use the rules of type (a) which make evolve the “counter” c, hence at each step we in crease by one the subscript of c. Therefore, when all variables are 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 object 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 a) propositional formula.

The previous example was chosen also for showing that the polarizations of membr anes are not used during generating the truth–assignments, but it might be useful after that – and, up to now, thi s is the case in all polynomial time solutions to **NP**-complete problems obtained in this framework, in part icular, for solving `SAT` (satisfiability of propositional formulas in the conjunctive normal form). This is an important *open problem* in this area: 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 handl ing the membrane structures. One of the early investigated additional possibility 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 labe l h, having inside the objects specified by v. Because we know the label of the new membrane, we know the rul es 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 operation s is also made visible.

For instance, converse to membrane division can be considered the operation of *merging* the contents of two membranes; 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, c are objects and h_1, h_2, h_3 are labels (we have considered the general case, where the labels can be changed).

Actually, the merging operation can be considered also as the reverse of the ıt 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 meanin g that the contents of membrane h_1 is split into two membranes, with labels h_2 and h_3, the first one co ntaining all objects from K and the second one containing all objects which are not in K.

Simple to formalize are also the operations of *endocytosys* and *exocyt osys* (we use these general names, although in biology there are distinctions depending on the size of the objects and the number of objects moved – phagocytosys, picocytosys, etc.).

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 endocytosys rule, stating that an elementary membrane labelled h_1 enters the adjacent membrane labelled

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 during the operation. Similarly, the rule $[_{h_2}[_{h_1}a]_{h_1}]_{h_2} \rightarrow [_{h_1}b]_{h_1}[_{h_2}\ ]_{h_2}$, for $h_1, h_2 \in H, a, b \in V$, indicates an exocytosys operation: an elementary membrane labelled h_1 is sent out of a membrane labelled 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. In the case of endocytosys, membrane h_2 can be a non-elementary one.

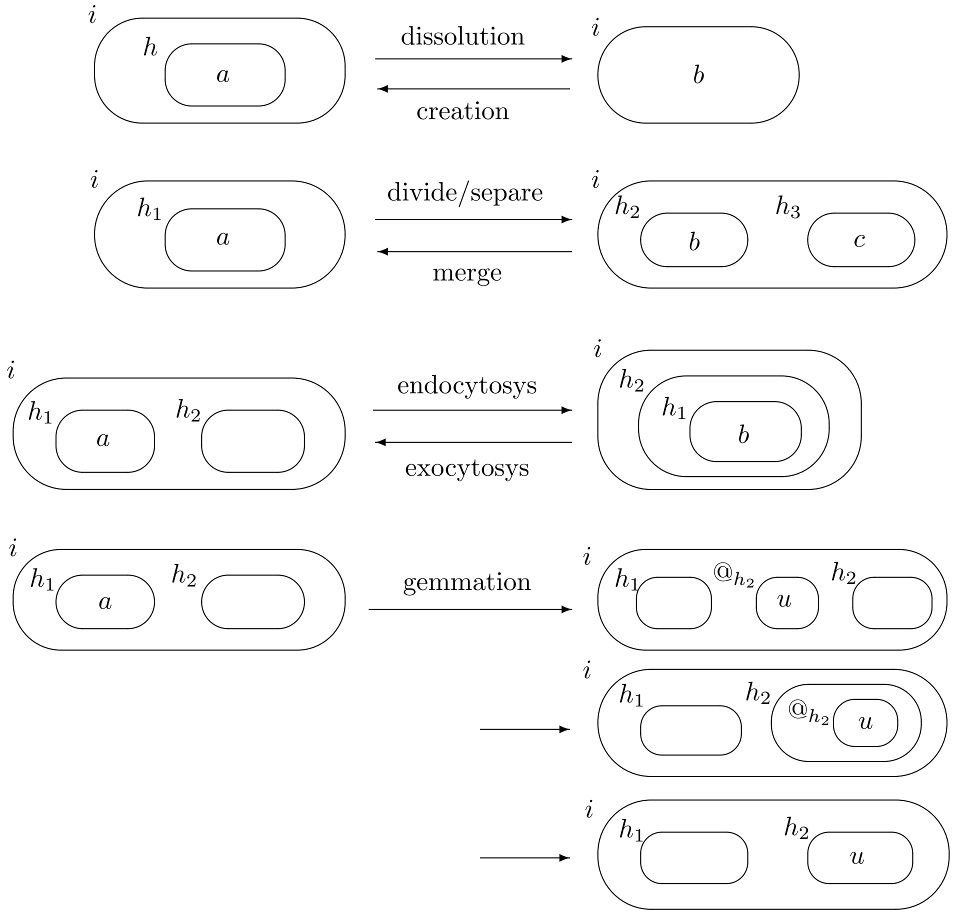


Figure 5: Membrane handling operations

Finally, let us mention the operation of *gemmation*, by which a membrane is created inside a membrane h_1 and sent to a membrane with label h_2; the moving membrane is dissolved inside the target membrane h_2 thus releasing its contents there. In this way, multisets of obje cts can be transported from a membrane to another one in a protected way: the enclosed objects cannot b e 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 dissol ved inside the membrane with label h_2. Corresponding to the situation from biology, in [17], [18] one considers only the case where the membranes h_1, h_2 are adjacent, and directly placed in the skin membrane, but the operation can be generalized.

Anyway, 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, with the important mentioning 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, [13], as well as in [57], and in other papers, but we do not ente r into further details here.

15 Structuring the Objects

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

Such a system has the form

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

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

We have given here the system in the 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 here we consider 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 ta rget indications *here, in, 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 w hether tar is *here, in*, or *out*, respectively. The strings which leave the system do not come back; if they are c omposed only of symbols from T, then they are considered as generated by the system, and included in the languag e $L(\Pi)$.

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 splicing rules as those from DNA computing (see [49], [85]), that is, of the form $u_1\#u_2\$u_3\#u_4$, where u_1, u_2, u_3, u_4 are strings over V. For four str ings $x, y, z, w \in V^*$ and a rule $r : u_1\#u_2\$u_3\#u_4$, we write

$$(x, y) \vdash_r (z, w) \quad \text{if and only if} \quad x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,$$

$$z = x_1u_1u_4y_2,\ w = y_1u_3u_2x_2,$$
$$\text{for some } x_1, x_2, y_1, y_2 \in V^*.$$

We say that we splice x, y at the sites u_1u_2, 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, 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, tar_2 one of *here, in, ou t.* 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 lan guage 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 pas s to introducing an important extension of rewriting rules, namely, *rewriting with replication*, [60]. 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 me aning that by rewriting a string x_1ax_1 we get n strings, $x_1u_1x_2, x_1u_2x_2, \dots, x_1u_nx_2$, whic h have to be moved in the regions indicated by targets $tar_1, tar_2, \dots, tar_n$, respectively. In this case we wo rk again with halting computations, and the motivation is that if we do not impose the halting conditi on, 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 $a \to (u_i, tar_i), 1 \le i \le n$, without changing the language; th at is, replication makes a difference only in the halting case.

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

Besides these types of rules for string processing, also other kinds of rules we re used, such as insertion and deletion, context adjoining in the sense of Marcus contextual grammars [76], 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 structur e, 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 establi shed among adjacent cells, by making their protein channels to cooperate, moving molecules directly from one cell to another cell, [64]). Actually, in the basic variant of tissue–like P sys tems, this graph is a virtually total one, what matters is the communication graph, dynamically define d during the computation. 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 is of the form $(i, x/y, j)$, where i, j are labels of cells or, at most one, is zero, identifyi ng the environment, and x, y are multisets of objects. The meaning is that the multiset x is moved from i to j, at the same time with moving the multiset y from j to i. If one of the multisets x, y is empty, then we ha ve, in fact, a symport rule. Therefore, the communication among cells is done either directly, in one step, o r indirectly, through the environment: one cell throws some objects out and other cells can ta ke 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 configurat ion and using the rules in the non-deterministic 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 wi th channel-states*, [41], 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, labelled 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, $i_o \in \{1, 2, ts, m\}$ 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 among 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, as well as (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 it (the system is synchronously evolving). 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, also vectors can be computed, by considering the multiplicity of objects from T present in cell i_o in the halting configurat ion.

A still more elaborated class of systems, called *population P systems*, were investigated in the last time in a series of papers by F. Bernardini and M. Gheorghe – see, e.g., [14] – with motivations related to the dynamics of cells in skin-like tissues, populations of bacteria, 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 these channels, can change during the evolution of the system, but also the cells can change their name, can disappear (get dissolved) and can divide, thus producing new cells; these new cells inherit, in a well specified sense, the links with the neighboring cells of the parent cell. The generality of this model makes it rather attractive for applications in areas 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 mor e complex cells, for instance, moving the states from the channels between cells to the cells the mselves – still preserving the network of synapses. This last term directly suggests the neural motivation of these attempts, aiming to capture something from the intricate structure of neural networks, of the way th e neurons are linked and cooperate in the most efficient computer ever invented, the human brain.

We do not recall here the formal definition of a neural-like P system, but we re fer to [82] for details, and here we only present 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 f inite set of states, a 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 stat es and w, x, y, z are multisets of objects; in state s, the cell consumes the multiset w and produces the multi sets x, y, z; the objects from multiset x remain in the cell, those of multiset y have to be communicated t o the cells towards 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, which is designated as the output cell, and for this cell, the u se of the previous rule entails sending the objects of z to the environment.

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

Because of the use of states, there are several possibilities of 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 maxim ally parallel manner: the multiset w is identified in the cell contents, in the maximal manner, and the rule is u sed 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 differen ce with the parallel mode, where in each step we choose a rule and we use only this rule as many times as possible).

Then, there also are three possibilities to move the objects between cells (of c ourse, we only move objects produced by rules in multisets with the indication *go*). Assume that we hav e applied a rule $sw \to s'(x, here)(y, go)$ in a given cell i. In the *spread* mode, the objects from y are non-deterministically distributed to all cells j such that (i, j) is a synapse of the system. In th e *one* mode, all the objects from y are sent to one cell j, again provided that the synapse (i, j) exist s. 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 – this is the *replicative* mode.

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 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 approach, grammar oriented, is only one possibility, mathematically attractive and important theoretically, but not useful from a practical point of view, when dealing with specific problems to solve and specific functions to compute. However, a P system can be used also 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 intro duce a multiset w_0 in a specified region, just adding the objects of w_0 to the objects present in tha t region. In the string case, a string can be added, possibly inserted in one of the existing strings. The com putation proceeds, and if it halts, then we say that the input is accepted (or recognized). In the transducer mode, we do not only have to halt, but we also collect an output, from a specified output region, internal to the s ystem or the environment.

Now, an important distinction appears, between systems which behave deterministically (in each moment, at most one transition is possible, hence either the computation stops, or it continues in a unique mode), and those which work in a non-deterministic way. Such a distinction does not makes much sense in the generative mode, especially if only halting computations provide a result, at their 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 non-deterministically, 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: yes or no. 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, important is that after n steps we get the same configuration. This brings into the stage the important notion of *confluence*. A system is *strongly confluent* if, starting from the initial configuration and behaving we-do-not-care-how, after a while it 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 confluence* property: irrespective how the system works, it always halts and all halting computations provide the same result. These notions will be essentially invoked when discussing the efficiency of P systems, as in Section 20.

Here let us consider in some details 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 [92], 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 [37] – see also [39]).

The above set $N_a(\Pi)$ was defined in general, for non-deterministic systems, but, clearly, in the accepting mode the determinism can be imposed (the non-determinism is moved to the environ ment, to the "user", which provides an input, unique, but non-deterministically chosen, from which the computation starts). Note that the example of a P system with sym port/antiport rules from Section 11 works in the same manner for an accepting register machine (a number is introduced in the first register and it 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 $l_1 : (\texttt{add}(r), l_2)$, with the continuation unique), and in this case the P system itself is deterministic.

19 Universality

The initial goal of membrane computing was to define computability models inspir ed from the cell biology, and indeed a large part of the investigations in this area was devoted to produc ing computing devices and examining their computing power, in comparison with the standard models in compu tability theory, Turing machines and their restricted variants. As it turns out, most of the considered classes o f P systems are equal in power with 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 from 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 un iversal one – the precise definition should be given for every particular type of systems). That is why we speak abou t *universality* results, and not about computational completeness.

All classes of systems considered above, whether cell-like, tissue-like, or neur al-like, with symbol-objects or string-objects, working in the generative or the accepting modes, of course, 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 which are 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, 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 a sufficiently large weight – see below stronger results from this point of view) .

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

As for notations, the family of sets $N(\Pi)$ of numbers (we keep from here the symbol N) generated by P systems of a specified type (we keep P), working with symbol-objects (O), having at most m membranes, and using features/ingredien ts from a given *list* is denoted by $NOP_m($*list-of-features*$)$. If we compute sets of vectors, then 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 w ith 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 memb ranes is not bounded, the subscript m is replaced by $*$, and this is a general convention, used also for other parame ters.

Now, in what concerns the list of features, they can be taken from an endless pool: using cooperative rules is indicated by *coo*, catalytic rules are indicated by *cat*, with the mentioning that the number of catalysts matters, hence we use cat_r in order to indicate that we use systems with at most r catalysts; bi-stable catalysts are indicated by $2cat$ ($2cat_r$, if at most r catalysts are used); similarly, 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*, endocytosys and exocytosys 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*, $repl_d$, *spl* for indicati ng that one uses rewriting rules, replicated rewriting rules (with at most d copies of each string produced by r eplication), 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 sympor t 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 uses to write nFe – for instance, $nPri$ 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 also 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 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 mentioned papers):

1. $NRE = NOP_1(cat_2)$, [38].
2. $NRE = NOP_3(sym_1, anti_1) = NOP_3(sym_2, anti_0)$, [4].
3. $NRE = NOP_3((a), (b), (c))$, [65].
4. $NRE = NOP_9(endo, exo)$, [59].
5. $NRE = NSP_3(repl_2)$, [61].

All results above hold true also for vectors of numbers.

In all these results, the number of membranes sufficient for obtaining the unive rsality is pretty small (the equality $NRE = NOP_9(endo, exo)$ was only recently proven and it is very prob able that it will be improved in the number of membranes). 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 no t modified), the hierarchy on the number of membranes collapses, because a number of membranes as large as the deg ree of the universal system suffices.

Still, "the number of membranes matters", as we read already in the title of įteIbarra: there are (sub-universal) classes of P systems for which the number of membranes induces an infinite hiera rchy of families of sets of numbers (see also [55]).

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*, t he resources used for solving problems. In general, the research in natural computi ng 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 a linea r time.

We have discussed above three basic ways to construct such an exponential space in cell-like P systems, namely, membrane division (the same effect has the separation operation, as well as other operations which replicate partially or totally the contents of a membrane), membrane creation (c ombined with the creation of exponentially many objects), and string replication. Similar possibilities are offered by cell division in tissue-like systems and by objects replication in neural-like systems. Also the possibility to use a pre-computed exponential work space, 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**-comple te problems were obtained. The first problem addressed in this context was `SAT` [81], [79] (the solution was improved in several respects in other subsequent papers), but similar solutions are reported in the literature for the Hamiltonian Path problem, the Node Covering problem, the problem of inverting one-way functions, the Subset-sum, a nd the Knapsack problems (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 [82], [87], as well as in the web page of the domain , [102].

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 c onstructed starting from a given problem, and an instance of the problem is introduced as a n input in such systems; working in a deterministic mode (or a *confluent* mode: som e non-determinism is allowed, provided that the branching converges after a while to a unique conf iguration, or, in the case of the 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 o bjects 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 a polynomial time. A more relaxed framework is that where a *semi-uniform* construction is allowed: carried out in polynomi al time by a Turing machine, but starting from the instance itself to be solved (the condition to have a poly nomial time construction ensures the "honesty" of the construction: the solution to the pro blem cannot be found during the construction phase).

This direction of research is very active at the present moment. More and more p roblems are considered, the membrane computing complexity classes are refined, characterizations of the **P**$\neq$**NP** conjecture were obtained in this framework, improvements are looked for. An important recent result concerns the fact that **PSPACE** was shown to be i ncluded 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 [92], [5] for details).

There also are many *open problems* in this area. We have mentioned already the intriguing question whether polynomial solutions to **NP**-complete problems can be obtained through P sys tems with active membranes without polarizations (and without label changing possibilities of other additio nal 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 divis ion cannot be avoided ("Milano theorem": a P system without membrane division can be simulated by a Turing mach ine with a polynomial slowdown, see [100], [101]).

21 Focusing on the Evolution

The computational power is of interest for theoretical computer science, computa tional efficiency is of interest for practical computer science, but none of these is of a direct interest for bi ology. Actually, this last statement is not at all correct: if a biologist is interested in simulating a cell – and this seems to become a major concern of to-day biology, see [58], [53] and other sources – t hen the generality of the model (its comparison with Turing machines and its restrictions) is directly linked to the possibility of solving algorithmically questions about the model. Just an example: is a given configura tion reachable from the initial configuration? Imagine that the initial configuration represents a healthy cell and we are interested whether a sickness state is ever reached. Then, if both healthy and non-healthy configur ations can be reached, the question appears 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 a nswer directly depends 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, i ndirectly, of interest also for biologists, so we (the biologists, too) should be more careful when asserting th at a given type of "theoretical" investigation is not of interest for biology.

Still, the immediate concern of biological research is the evolution of biological systems, their life, whatever this means, not the result of a specific evolution. Otherwise stated, halting computations are of interest for computer science, of direct interest for 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 towards 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 were imagined for making also the membrane structure 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 with 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 [19], [68], [67], [15], etc. 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 befor e – after just repeating the fact that topics such as complexity classes and polynomial solutions to hard problems, dyn amical systems approaches, 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 significan t theoretical and practical results. To these trends we can add another general, and not very structured yet, topic: using non-crisp mathematics, handling uncertainty by means of probabilistic, fuzzy sets, rough sets 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 syste ms where there exists no internal clock, and all rules have different times to get applied? This can mean both tha t the duration needed by a rule to get applied can differ from the duration of another rule, and, the extreme possibili ty, that the duration is not known at all. In the first case, we can have a timing function, assigning durations to rules, in the second case even such an information is missing (e.g., a rule $a \to v$ should wait for a rul e $c \to ba$ to be completed in order to have an available a to process). How the power of a system depends on the timi ng function? Are there time-free systems, which generate the same set of numbers irrespective which is the time f unction which associates durations with its rules? Such questions are addressed in a series of papers by M. Cavali ere and D. Sburlan; see e.g., [27], [28].

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 a large computing power, with an *observer* Obs, a finite state machine which analyzes the configurations of the system Sys along the evolutions of the system; from each configuration either a symbol is produced or nothing (that is, the "result" of that configuration is the empty string λ); 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 [24], [25].

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 "orthogona l" on the usual questions concerning the number of membranes and the size of rules, as, intuitively, one o f these parameters should be left free in order to codify and handle an arbitrary amount of information by using a limi ted number of objects. The first results of this type were given in [84] 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 com puted by so-called (see [46]) 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 t he sets of vectors they accept can be non-semilinear).

Other interesting topics recently investigated which we only list here concern t he reversibility of computations in P systems [63], energy accounting (associating quanta of energy to objects or to rules, handled during the computation) [43], [42], [62], relations with grammar systems and w ith colonies [83], descriptional complexity, non-discrete multisets [74], [34].

We close this section by mentioning the notion of the *Sevilla carpet* intro duced in [31], 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. Thi s corresponds to the Szilard language from language theory, with the complication now that we use several rul es in the same step, and each rule is used several times. Considering all the information concerning the rules we c an get a global evaluation of the complexity of a computation – as nicely illustrated, for instance, in [90] and [47].

23 Applications; The Attractiveness of Membrane Computing as a Modelling F ramework

Finally, let us shortly discuss some applications of membrane computing – start ing however with a general discussion about the features of this area of research which make it attractive for applications in several disciplines, especially for biology.

First, there are several keywords which are genuinely proper to membrane computing and which are of interest for many applications: *distribution* (with the important system-part interaction, emergent behavior, non-linearly resulting from the composition of local behaviors), *discrete mathematics* (continuous mathematics, especially systems of differential equations, has a glorious history of applications in many disciplines, such as astronomy, physics, meteorology, but it failed to prove adequate for linguistics, and cannot cover more than local processes in biology because of the complexity of processes and, in many cases, their imprecise character; then, a basic question is whether the biological reality is of a continuous nature or of a discrete nature – as languages proved to be, which th e second possibility ruling out the usefulness of many tools from continuous mathematics), *algorithmicity* (by definition, P systems are computability models, of the same type as Turing machines or other classic representations of algorithms, hence easy to be simulated on a computer), *scalability/extensibility* (this is one of the main difficulties of using differential equations in biology), *transparency* (multiset rewriting rules are nothing else than reaction equations as customarily used in chemistry and bio-chemistry, without any "mysterious" notation and, behind the notation, "mysterious" behavior), *parallelism* (a dream of computer science, a common sense in biology), *non-determinism* (let us compare the "program" of a P system, which is a set of instructions/rules, with the only structure being that imposed by localization to regions, but without any ordering/structure inside each region, with the rigid sequences of instructions of programs written in usual programming languages), *communication* (with the marvellous and still not completely understood way the life is coordinating the many processes taking place in a cell, and in tissues, organs, organisms, in contrast with the costly way of coordinating/synchronizing computations in parallel electronic computing architectures, where the communication time become prohibitive with the increase of the number of processors), and so on and so forth.

Then, for biology, besides the easy understanding of the formalism and the trans parency of the (graphical and symbolic) representations, encouraging should be also the simple observation tha t membrane computing emerged as a bio-inspired research area, explicitly looking to the cell for finding comp utability models (though, not looking initially for models of relevance for the biological research), hence it is just natural to try to improve these models and use them in the study of the very originating ground. This should be put in contrast with the attempt to "force" models and tools developed in other scientific areas, e.g., in physi cs, to cover biological facts, presumably of a genuinely different nature as that of the area for which these models and t ools were created and proven to be adequate/useful.

Coming back to the important distinction between continuous and discrete tools, it should be emphasized that significant results can be obtained by computer simulations of discrete data, in particular, of multisets. This has been convincingly proven in many cases – see [94], [93], [95], [74], [68], [67] – and reminds of the assertion made in several places that cellular automata can be an alternative/substitute for differential equations; comparing the rigid structure of cellular automata with the flexibility of membrane systems we can safely infer that if this assertion is va lid for cellular automata, then it is still "more valid" for P systems.

Now, in what concerns the applications themselves reported up to now, they are d eveloped at various levels. In many cases, what is actually used is the *language* of membrane computing , having in mind three dimensions of this aspect: (i) the long list of concepts either newly intr oduced, or related in a new manner in this area, (ii) the mathematical formalism of membrane computing, and (iii) t he graphical language, the way to represent membranes, cell-like structures, tissue-like structures. It is easy to illustrate each of these three points, e.g., producing a list of concepts already mentioned in the prese nt paper, or recalling some of the stabilized notations (for instance, representing a membrane by square brackets, with labels and possibly electrical charges, with the multiset rewriting rules, the symport and antiport rules), but we only say some words about the graphical language. Many ingredients are, in a great extent, known: Eu ler–Wenn diagrams (here, without intersection and with a unique superset, corresponding to the skin membrane), wi th labels assigned to membranes, with multisets (not sets!) of objects placed inside, with arrows desc ribing the communication channels in the case of tissue-like and neural-like systems; what is essentially new is t hat also the rules for the evolution of the system are written in compartments in the case of multiset or string rewr iting rules, and near membranes in the case of symport/antiport and boundary rules (which are associated with me mbranes). Thus, not only the state of the system is displayed, but also the "evolution engine", the rules. Also, t he localization is apparent, both for objects and rule.

However, this level of application/usefulness is only a preliminary, superficial one. The next level is to use tools, techniques, results of membrane computing, and here there appears an important question: to which aim? Solving problems already stated, e.g., by biologists, in other terms and another framework, coul d be an impressive achievement, and this is the most natural way to proceed – but not necessarily the most efficient one, at least at the beginning. New tools can suggest new problems, which either cann ot be formulated in a previous framework (in plain language, as it is the case in biology, whatever s pecialized the specific jargon is, or using other tools, such as differential equations) or have no chan ce to be solved in the previous framework. Problems of the first type (already examined by biologists, mainly experimentall y) concern, for instance, correlations of processes, of the presence/absence of certain chemicals, their m ultiplicity (concentration, population) in a given compartment, their interaction, while of the second type are topics related to the trajectories of bio-systems when modelled as dynamical systems (e.g., a sequence of configuratio ns can be finite or infinite, while in the latter case it can be periodic, ultimately periodic, almost periodi c, quasi-periodic, etc., notions which are not yet present in the index of notions of biological books).

Applications of all these types were reported in the literature of membrane comp uting. As expected and as natural, most applications were carried out in biology, but also applications in computer graphics (where the compartmentalization seems to add a significant efficiency to well-known techniq ues based on L systems, [44]), linguistics (both as a representation language for various concep ts related to language evolution, dialogue, semantics [13], and making use of the parallelism, in solving parsing problems in an efficient way [45]), management (again, mainly at the l evel of the formalism and the graphical language, see, e.g., [11], [12]), in devising sortin g and ranking algorithms [7], handling 2D structures [29], etc.

In turn, the applications in biology follow in most cases a scenario of the following type: one examines a piece of reality, in general from the biochemistry of the cell, one writes a P system modelling the respective process, one writes a program simulating that system (or one uses one of the existing programs), and one performs a large number of experiments with the program (this is much cheaper than conducting laboratory experiments), tuning certain parameters, and looking for the evolution of the system (usually, for the population of certain objects). We do not recall any detail here, but we refer to the chapter of [82] devoted to biological applications, as well as to the papers available in the web page [102], and, especially, to the forthcoming volume [30]. Anyway, the investigations are somewhat preliminary, but the progresses are obvious and the hope is to have in the near future applications of an increased interest for biologists.

This hope is supported also by the fact that more and more powerful simulations/implementations of various classes of P systems are available, with better and better interfaces, which allow for the friendly interaction with the program. We avoid to plainly s ay that we have "implementations" of P systems, because of the inherent non-determinism and the massive parallelis m of the basic model, features which cannot be implemented, at least in principle, on the usual electronic computer – but which can be implemented on a dedicated , reconfigurable, hardware, as done in [88], or on a local network, as reported in [32] and [96]. This does not mean that simulations of P systems on usual comp uters are not useful; actually, such programs were used in all biological applications mentioned above , and can also have important didactic and research applications. An overview of membrane computing software reported in literature (some programs are available in the web page [102]) can be found in [48].

24 Closing Remarks

The present paper should be seen only as a general overview of membrane computin g, with the choice of topics intended to be as pertinent as possible, but, of course, not completely f ree of a subjective bias. The reader interested in further technical details, formal definitions, proofs, research topics and open problems, or in details concerning the applications (and the software behind them) is advised to consult the comprehensive web page from `http://psystems.disco.unimib.it`. A complete bibliography of membrane computing can be found there, with many papers availabl e for downloading (in particular, one can find there the proceedings volumes of the yearly Workshops on Membrane C omputing, 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 [72], 147–161.

[4] A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan : Communicative P Systems with Minimal Cooperation. In [72], 162–178.

[5] A. Alhazov, C. Martín-Vide, L. Pan: Solving a PSPACE-Compl ete 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-Pro ceedings of Workshop on Membrane Computing*, WMC 2003, Tarragona, Spain, July 2003. Technical Report 28/03, Rovira i Virgili University, Tarragona, 2003.

[7] A. Alhazov, D. Sburlan: Static Sorting Algorithms for P Systems. In [70], 17–40.

[8] I.I. Ardelean: The Relevance of Biomembranes for P Systems. *Fundamenta Informaticae*, 49, 1–3 (2002), 35–43.

[9] 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.

[10] J.-P. Banâtre, P. Fradet, D. Le Métayer: Gamma and the Chem ical Reaction Model: Fifteen Years After. In [21], 17–44.

[11] J. Bartosik: Paun's Systems in Modeling of Human Resource Manag ement. *Proc. Second Conf. Tools and Methods of Data Transformation*, WSU Kielce, 2004.

[12] J. Bartosik, W. Korczynski: Systemy membranowe jako modele hier archicznych struktur zarzadzania. *Mat. Pokonferencyjne Ekonomia, Informatyka, Zarzadzanie. Teori a i Praktyka*, Wydzial Zarzadzania AGH, Tom II, AGH 2002.

[13] G. Bel Enguix, M.D. Jiménez-Lopez: Linguistic Membrane Systems and Applications. In [30].

[14] F. Bernardini, M. Gheorghe: Population P Systems. *Journal of Universal Computer Science*, 10, 5 (2004), 509–539.

[15] F. Bernardini, V. Manca: Dynamical Aspects of P Systems. *Bi oSystems*, 70, 2 (2003), 85–93.

[16] G. Berry, G. Boudol: The Chemical Abstract Machine. *Theoreti cal Computer Science*, 96 (1992), 217–248.

[17] D. Besozzi: *Computational and Modelling Power of P System s.* PhD Thesis, Univ. degli Studi di Milano, 2004.

[18] D. Besozzi, C. Zandron, G. Mauri, N. Sabadini: P Systems with Gem mation of Mobile Membranes. *Proc. ICTCS 2001*, Torino, *LNCS* 2202 (A. Restivo, S.R. Della Rocca, L. Roversi, eds.), Springer-Verlag, Berlin, 2001, 136–153.

[19] C. Bonanno, V. Manca: Discrete Dynamics in Biological Models. *Romanian Journal of Information Science and Technology*, 5, 1-2 (2002), 45–67.

[20] C. Calude, Gh. Păun: Bio-Steps Beyond Turing. *BioSystem s*, 2004.

[21] 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.

[22] L. Cardelli: Brane Calculus. *Proc. Computational Methods in Systems Biology '04*, Springer, to appear.

[23] M. Cavaliere: Evolution-Communication P Systems. In [86], 134–145.

[24] M. Cavaliere, P. Leupold: Evolution and Observation – A New Way to Look at Membrane Systems. In [70], 70–87.

[25] M. Cavaliere, P. Leupold: Evolution and Observation. A Non-Standa rd Way to Generate Formal Languages. *Theoretical Computer Science*, 321, 2-3 (2004), 233–248.

[26] M. Cavaliere, C. Martin-Vide, Gh. Păun, eds.: *Proceedin gs of the Brainstorming Week on Membrane Computing, Tarragona, February 2003.* Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003.

[27] M. Cavaliere, D. Sburlan: Time-Independent P Systems. In [72], 239–258.

[28] M. Cavaliere, D. Sburlan: Time and Synchronization in Membrane Systems. *Fundamenta Informaticae*, 64 (2005), 65–77.

[29] R. Ceterchi, R. Gramatovici, N. Jonoska, K.G. Subramanian: Generati ng Picture Languages with P Systems. In [26], 85–100.

[30] G. Ciobanu, Gh. Păun, M.J. Pérez–Jiménez, eds.: *Applications of Membrane Computing.* Springer, Berlin, 2005.

[31] G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla Carpets Asso ciated with P Systems. In [26], 135–140.

[32] G. Ciobanu, G. Wenyuan. A P System Running on a Cluster of Computers. In [70], 123–139.

[33] L. Colson, N. Jonoska, M. Margenstern: λP Systems and T yped λ-Calculus. In [72], 1–18.

[34] A. Cordón-Franco, F. Sancho-Caparrini: Approximating Non-Disc rete P Systems. In [72], 288–296.

[35] S. Crespi–Reghizzi, D. Mandrioli: Commutative Grammars. *Cal colo*, 13, 2 (1976), 173–189.

[36] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, G. Vaszil: Cells in Environment: P Colonies. Submitted, 2004.

[37] E. Csuhaj-Varju, G. Vaszil: P Automata or Purely Communicating Ac cepting P Systems. In [86], 219–233.

[38] R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally Universa l P Systems Without Priorities: Two Catalysts Suffice. *Theoretical Computer Science*, 2004.

[39] R. Freund, M. Oswald: A Short Note on Analysing P Systems. it Bulletin of the EATCS, 78 (2003), 231–236.

[40] R. Freund, A. Păun: Membrane Systems with Symport/Antiport Rules: Universality Results. In [86], 270–287.

[41] 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*, 2004, in press.

[42] P. Frisco: *Theory of Molecular Computing. Splicing and Mem brane Systems.* PhD Thesis, Leiden University, The Netherlands, 2004.

[43] P. Frisco, S. Ji: Towards a Hierarchy of Info-Energy P Systems. I n [86], 302–318.

[44] A. Georgiou, M. Gheorghe, F. Bernardini: Generative Devices Used in Graphics. In [30].

[45] R. Gramatovici, G. Bel Enguix: Parsing with P Automata. In [30].

[46] S.A. Greibach: Remarks on Blind and Partially Blind One-Way Multi counter Machines. *Theoretical Computer Science*, 7 (1978), 311–324.

[47] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: On Descriptive Complexity of P Systems. In [72], 321–331.

[48] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Nú nez. Available Membrane Computing Software. In [30].

[49] T. Head: Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors. *Bulletin of Mathemat ical Biology*, 49 (1987), 737–759.

[50] J. Hartmanis: About the Nature of Computer Science. *Bulletin of the EATCS*, 53 (June 1994), 170–190.

[51] J. Hartmanis: On the Weight of Computation. *Bulletin of the EATCS*, 55 (Febr. 1995), 136–138.

[52] J. Hoffmeyer: Surfaces Inside Surfaces. On the Origin of Age ncy and Life. *Cybernetics and Human Knowing*, 5, 1 (1998), 33–42.

[53] M. Holcombe: Computational Models of Cells and Tissues: Machines, Agents and Fungal Infection. *Briefings in Bioinformatics*, 2, 3 (2001), 271–278.

[54] O.H. Ibarra: The Number of Membranes Matters. In [70], 218–231.

[55] O.H. Ibarra: On Membrane Hierarchy in P Systems. *Theore tical Computer Science*, 2004.

[56] O.H. Ibarra: On Determinism Versus Nondeterminism in P Syste ms. Submitted, 2004.

[57] M. Ionescu, T.-O. Ishdorj: Replicative–Distribution Rules in P S ystems with Active Membranes. *Proc. of ICTAC2004, First Intern. Colloq. on Theoretical Aspects of Computing*, Guiyan g, China, 2004.

[58] H. Kitano: Computational Systems Biology. *Nature*, 420, 14 (2002), 206–210.

[59] S.N. Krishna, Gh. Păun, P Systems with Mobile Membranes. it Theoretical Computer Science, 2005.

[60] S.N. Krishna, R. Rama: P Systems with Replicated Rewriting. *Journal of Automata, Languages and Combinatorics*, 6, 3 (2001), 345–350.

[61] S.N. Krishna, R. Rama, H. Ramesh: Further Results on Contextua l and Rewriting P Systems. *Fundamenta Informaticae*, 64 (2005), 235–246.

[62] 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.

[63] A. Leporati, C. Zandron, G. Mauri: Universal Families of Revers ible P Systems. *Proc. Conf. Universal Machines and Computations 2004*, Sankt Petersburg, 2 004.

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

[65] M. Madhu, K. Krithivasan: Improved Results About the Universality o f P Systems. *Bulletin of the EATCS*, 76 (Febr. 2002), 162–168.

[66] V. Manca: String Rewriting and Metabolism. A Logical Perspective . In Gh. Păun, ed.: *Computing with Bio-Molecules. Theory and Experiments*, Springer, Singapore, 1998, 36–60.

[67] V. Manca, L. Bianco, F. Fontana: Evolution and Oscillation in P Systems: Applications to Biological Phenomena. In [72], 63–84.

[68] V. Manca, G. Franco, G. Scollo: State Transition Dynamics. Ba sic Concepts and Molecular Computing Perspectives. In M. Gheorghe. ed.: *Molecular Computational Models . Unconventional Approaches*, Idea Group, London, 2004.

[69] S. Marcus: Bridging P Systems and Genomics: A Preliminary Appr oach. In [86], 371–376.

[70] C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. S alomaa, eds.: *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised P apers. Lecture Notes in Computer Science*, 2933, Springer, Berlin, 2004.

[71] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Pat ón: Tissue P Systems. *Theoretical Computer Science*, 296, 2 (2003), 295–326.

[72] G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg , A. Salomaa, eds.: *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revise d Papers, Lecture Notes in Computer Science*, 3365, Springer, Berlin, 2005.

[73] M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.

[74] T.Y. Nishida: Simulations of Photosynthesis by a K-subset Tran sforming System with Membranes. *Fundamenta Informaticae*, 49, 1-3 (2002), 249–259.

[75] A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, 20, 3 (2002), 295–306.

[76] Gh. Păun: *Marcus Contextual Grammars.* Kluwer, Dordrech t, 1997.

[77] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Comp uter Science-TUCS Report 208, November 1998, `www.tucs.fi`).

[78] Gh. Păun: Computing with Membranes – A Variant. *International Journal of Foundations of Computer Science*, 11, 1 (2000), 167–1 82.

[79] Gh. Păun: Computing with Membranes: Attacking NP-Complete Prob lems. In I. Antoniou, C.S. Calude, M.J. Dinneen, eds.: *Unconventional Models of Com putation*, Springer, London, 2000, 94–115.

[80] Gh. Păun: From Cells to Computers: Computing with Membrane s (P Systems). *BioSystems*, 59, 3 (2001), 139–158.

[81] Gh. Păun: P Systems with Active Membranes: Attacking NP-Compl ete Problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 7 5–90.

[82] Gh. Păun: *Computing with Membranes: An Introduction.* S pringer, Berlin, 2002.

[83] Gh. Păun: Grammar Systems vs. Membrane Computing: A Prelimina ry Approach. *Workshop on Grammar Systems*, MTA SZTAKI, Budapest, 2004, 225–245.

[84] Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-P atón: Symport/Antiport P Systems with Three Objects Are Universal. *Fundamenta Inf ormaticae*, 64 (2005), 345–358.

[85] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New C omputing Paradigms.* Springer, Berlin, 1998.

[86] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.: *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, R omania, Revised Papers. Lecture Notes in Computer Science*, 2597, Springer, Berlin, 2003.

[87] M. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas.* Editorial Kronos, Sevilla, 200 2.

[88] B. Petreska, C. Teuscher: A Hardware Membrane System. In [70], 269–285.

[89] A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro: B ioAmbients – An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325 (2004), 141–16 7.

[90] A. Riscos–Núñez: *Programacion celular. Resolucion ef iciente de problemas numericos NP-complete.* PhD Thesis, Univ. Sevilla, 2004.

[91] P. Sosik: The Computational Power of Cell Division in P Systems : Beating Down Parallel Computers? *Natural Computing*, 2, 3 (2003), 287–298.

[92] P. Sosik, J. Matysek: Membrane Computing: When Communication Is Enough. In C.S. Calude, M.J. Dinneen, F. Peper, eds., *Unconventional Models of Computation 2002, Lecture Notes in Computer Scienc e*, 2509, Springer, Berlin, 2002, 264–275.

[93] Y. Suzuki, Y. Fujiwara, H. Tanaka, J. Takabayashi: Artificial Life Applications of a Class of P Systems: Abstract Rewriting Systems on Multisets. In [21], 299–346.

[94] Y. Suzuki, H. Tanaka: Chemical Oscillation in Symbolic Chemica l Systems and Its Behavioral Pattern. In Y. Bar-Ylam, ed.: *Proc. Intern. Conference on Complex Systems*, New England Complex Systems Institute, 1997, 1–7.

[95] Y. Suzuki, H. Tanaka: Abstract Rewriting Systems on Multisets, and Its Application for Modelling Complex Behaviours. In [26], 313–331.

[96] A. Syropoulos, P.C. Allilomes, E.G. Mamatas, K.T. Sotiriades: A Distributed Simulation of P Systems. In [70], 355–366.

[97] C. Teuscher: *Alan Turing. Life and Legacy of a Great Thinker* . Springer, Berlin, 2003.

[98] M. Tomita: Whole-Cell Simulation: A Grand Challenge of the 21st Century. *Trends in Biotechnology*, 19 (2001), 205–210.

[99] G. Vaszil: On the Size of P Systems with Minimal Symport/Antipo rt. *Pre-Proceedings of Workshop on Membrane Computing, WMC5, Milano, Italy*, June 2004, 422–431.

[100] C. Zandron: *A Model for Molecular Computing: Membrane S ystems.* PhD Thesis, Univ. degli Studi di Milano, 2001.

[101] C. Zandron, C. Ferretti, G. Mauri: Solving NP-Complete Probl ems Using P Systems with Active Membranes. In I. Antoniou, C.S. Calude, M.J. Dinneen, eds. : *Unconventional Models of Computation*, Springer, London, 2000, 289–301.

[102] * * * The Web Page of Membrane Computing: `http://psystems.disc o.unimib.it`