Computing with cells: membrane systems – some complexity issues

Oscar H. Ibarra and Andrei Păun

Department of Computer Science, University of California, Santa Barbara, CA 93106, USA; National Institute for Research and Development for Biological Sciences, Bucharest, Romania; Facultad de Informática Campus de Montegancedo S/N, Universidad Politécnica de Madrid – UPM, Boadilla del Monte, 28660 Madrid, Spain; Department of Computer Science/IfM, Louisiana Tech University, Ruston, LA 71272, USA

Membrane computing is a branch of natural computing which abstracts computing models from the structure and the functioning of the living cell. The main ingredients of membrane systems, called P systems, are (i) the membrane structure, which consists of a hierarchical arrangements of membranes which delimit compartments where (ii) multisets of symbols, called objects, evolve according to (iii) sets of rules which are localised and associated with compartments. By using the rules in a nondeterministic/deterministic maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions is a computation of how the system is evolving. Various ways of controlling the transfer of objects from one membrane to another and applying the rules, as well as possibilities to dissolve, divide or create membranes have been studied. Membrane systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems once future biotechnology gives way to a practical bio-realization. In this paper we survey some interesting and fundamental complexity issues such as universality vs. nonuniversality, determinism vs. nondeterminism, membrane and alphabet size hierarchies, characterizations of context-sensitive languages and other language classes and various notions of parallelism.

Keywords: membrane system; symport/antiport system; catalytic system; context-sensitive language

1. Introduction

The relationship between Computer Science, ranging from very theoretical to very practical, and Biology is quite fascinating: each domain has helped, even influenced, the other domain in an essential manner. In Biology one can mention only the Human Genome Project or the data mining techniques used for identifying genes or groups of genes responsible for different diseases. On the other hand, in Computer Science, many important classes of models found their inspiration in biology. Examples include finite automata and their link to neural nets (McCulloch, Pitts, Kleene) where the roots of the modern neural networks area are also found, cellular automata (Conway, Gardner), genetic algorithms and, more generally, evolutionary computing/programming, with their declared intention to mimic evolution at the genetic level. Several other biologically inspired domains in Computer Science are recently very active; one such domain is for example the area of *membrane computing* that was recently selected by the Institute for Scientific Information (ISI) as a fast 'Emerging Research Front' in Computer Science. See the article at the website: http://esi-topics.com/erf/october2003.html.

The cell, the building block of the organisms is an extremely complicated system highly adaptive with an unbelievable degree of self-configuration and self-maintenance. The membrane computing area was started with the ambitious goal to perform computations using these biological building blocks, the cells. Its goal is to abstract a computing model from the structure and functioning of the living cell. Initiated only few years ago, the domain has been vividly investigated, mainly from a mathematical point of view. There are many types of membrane systems, also called P systems, that are computationally complete. Many are able to solve hard problems in polynomial time by making use of an exponential space created in a natural way, for instance, by cell division, or string replication. No experiment of computing in a cell has yet been reported, but the domain is continuously growing, and is currently trying to return to the originating area, biology. Membrane systems can be a possible step in the 'main post-genomic task: modelling/simulating the living cell' (Holcombe, Tomita).

We present here a succinct overview of the membrane systems area touching briefly on topics related to universality vs. nonuniversality, determinism vs. nondeterminism, hierarchies, characterizations context-sensitive languages and other language classes and parallelism vs. sequentiality of such systems. We refer the interested reader

for in-depth descriptions, proofs and the many other topics that were not covered in this survey.

2. Membrane systems: features and an informal description

In short, membrane computing abstracts computing models, starting from the structure and functioning of living cells and continuing with their organization in tissues, organs, etc. These models are distributed and parallel computing models, processing multisets of symbol objects in a localised manner. The evolution rules and evolving objects are encapsulated into compartments delimited by membranes. An essential feature of these systems is the ability to communicate among compartments or between compartments and environment.

The essential ingredients of a membrane system are its *membranes*, which delimit compartments where *multisets* of objects evolve according to specific *evolution rules*. The membranes in our framework correspond to the biological lipidic membranes encountered in all cells. Such a membrane structure is a set of (labelled) membranes arranged hierarchically, where the whole hierarchy is contained in a distinguished external membrane corresponding to the plasma membrane and usually called the *skin* membrane. Each membrane determines a compartment, also called a *region*, which is the space delimited from above by the membrane itself and from below by the membranes placed directly inside, if any exist. Clearly, the correspondence membrane-region is one-to-one, that is why these terms are used interchangeably in what follows. We will be using the labels of membranes for identifying the regions they delimit. In all these regions delimited by the membranes we have objects modelling the inner molecules of a cell and localised rules applicable on the objects modelling the interactions between the molecules and their transformations in cells. Because of their general definition and, implicitly, their flexibility in modelling any type of molecule at the level of the cell, one could chose to model the processes related to the cells and membranes at different levels and orders of magnitude, for example going from high to low, one can model interactions between cells as it was done or model interactions between the cell and its environment as it was done or consider only the inside of the cell and model processes that take place there

Inspired by the biochemistry taking place in the cells, the evolution rules are used in general in parallel in a nondeterministic manner, but several other possibilities were also considered recently.

It is important to note that several features important for computer science were touched upon – distribution, parallelism, communication, synchronization, decentralization, nondeterminism – with a specific definition and materialization, as suggested by the biological reality.

Now, having in mind the previous discussion, let us make things a bit more concrete by considering a particular variant. Let us assume that we have one membrane, delimiting the outside environment from the cell that contains a multiset of objects that evolve according to specific rules. The multisets from the compartments of our computing device are processed by rewriting-like rules. Note that in our simple example we have only two regions/compartments. This means that rules are of the form $u \longrightarrow v$, where u and v are multisets of objects, represented by strings. As an example, consider the rule $abb \longrightarrow accddd$. It indicates the fact that one copy of object a together with two copies of object b react, and, as a result of this reaction, we get back a copy of a (and in this rule a behaves as a catalyst), and we produce also two copies of c and three copies of d. If this rule is applied to the multiset $a^2b^5c^4d$, then, because abb are 'consumed' and then $accddd$ are then 'produced' by the rule aforementioned, we pass to the configuration $a^2b^3c^6d^4$ by applying the previous rule once. Similarly, by using in this simple system a second rule: $aa \longrightarrow bbc$, we will get the configuration $b^5c^7d^4$, which contains no occurrence of object a, thus no further 'rewriting' is possible.

We now discuss two important notions: determinism or lack thereof, and parallelism which are very important for the membrane systems area and also fundamental for computer science in general. Addressing problems in a nondeterministic way, i.e. guessing a solution and then checking it, is a fast way to solve a problem, but it is not very viable from a practical point of view, because the guessing part can be wrong and then the user will not get the desired output. A solution for this problem can be given by the parallelism, which can help by simulating at the same time all branches of the nondeterminism and, thus, provide the desired solution. This is viable especially in the case of bio-chemical-like frameworks, where reactions take place in a massively parallel way. In a sense this brings the membrane computing area very close to the DNA computing by harnessing the power of parallelism in these devices to supplement for the 'awkward' (from a Computer Science point of view) basic operations that the cell performs *in vivo*. The combination between nondeterminism and parallelism represent the core features needed for the realization of promise of these systems that someday they could be solving problems that are otherwise intractable by the 'regular' computers.

Thus in the basic models of membrane computing both nondeterminism and parallelism are present. The rules to be used and the object(s) to which they are applied are chosen randomly, while in a given step one applies in parallel as many rules as possible. One could say that the parallelism is maximal – and this was the way the evolution rules were used in and in most papers in membrane computing dealing with theoretical issues.

An important observation that needs to be made is that we are speaking about synchronised devices, in the sense that a global clock is assumed, marking the time in the same way for all compartments of the system. This feature is apparent in the natural setting when one considers for example the beating of the heart, it is obvious that in that tissue, all the cells need to contract at specific time-steps to achieve the function, so it is clear that at least in that tissue we can see the synchronization of the cells. Another example for such a behaviour is the synchronization of the neurons that is achieved through the inhibitory properties of GABA in networks of neurons, etc. Let us now go back to our example; the use of a rule takes one time unit, and in each time unit all objects which can evolve should do it. More precisely, in each time unit, the multiset from each compartment must evolve by means of a maximal multiset of applicable rules. We illustrate this idea with the previous multiset and the pair of rules defined. Using these rules in the maximally parallel manner means to either use the first rule twice and thus involving both copies of a and four copies of b, or the second rule once – only once because it consumes both copies of a, and in this way the first rule cannot be used at the same time with the second rule. In the first case, one copy of b remains unused and the resulting multiset is $a^2b^1c^8d^7$; in the second case, all copies of b and c remain unused, and the resulting multiset is b^7c^5d. We note that in the second case the maximally parallel application of rules corresponds a *sequential* (one at a time) application of the second rule.

More specifically, the nondeterministic maximally parallel manner of application for the rules means that we assign objects to rules, nondeterministically choosing the objects and the rules, until no further assignment is possible due to the fact that we 'consumed' all the objects that could 'react'. More mathematically stated, we look at 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 as many times as indicated by their multiplicities and (ii) the multiset is *maximal*, meaning no further rule can be added to it because of the lack of available objects.

Thus, an evolution step in a given region consists of finding a maximal applicable multiset of rules, removing from the region all objects specified in the left hand 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 rules. If at least one of the rules introduces the dissolving action δ, then the membrane is dissolved, and its contents becomes part of the immediately upper membrane – provided that this membrane was not dissolved at the same time, a case where we stop in the first upper membrane which was not dissolved. Note that at least the skin membrane has to remain intact.

Using the rules as suggested above, one obtains transitions between configurations of a membrane system. A sequence of transitions forms a computation, and with a halting computation, i.e. one reaching a configuration where no rule can be applied, we associate a result. Because in the system we have numbers (multiplicities of objects), the result will be a number, e.g. the total number of objects in a specified compartment, or a vectors of numbers, in the case where we distinguish among the objects from that compartment. In this way, we get a number (or vector) generating device.

There are many variants of the architecture and functioning of the computing device sketched above, and we will mention some of them after introducing in some detail the basic type of membrane systems, the cell-like one working with symbol objects processed by communication-only rules.

3. A membrane system variant

Following the discussion from the previous section, we will describe a basic variant for membrane systems. In the following we will give the formal definition for a system that works on symbols, and with evolution rules modelled from the way the molecules move through the lipidic membranes of cells.

In our context, the rules as exemplified in the previous section can be changed in various ways: using the targets here, out, in, in$_j$ with the indication of the label of the target membrane; allowing catalysts to move across membranes; using 'pseudo-catalysts', which are allowed to flip from a state c to a state c' and back; having rules that result in membrane dissolving, i.e. a special symbol is generated in a membrane that is then dissolved; using an operation opposite to membrane dissolving, such as membrane thickening, i.e. making it nonpermeable to object communication, or even creating new membranes. Similarly, one can control the use of rules, e.g. by means of promoters, i.e. objects which must be present in order to apply a rule, but which are not modified by a rule and which can promote at the same time several rules – note that the promoters do not inhibit the parallelism, or inhibitors, by considering a priority relation among rules, etc. Note that these controls decrease the degree of nondeterminism of the system. Out of all these possibilities we chose to describe the membrane systems based on symport/antiport.

This model represents an important class of membrane systems and is that based on symport/antiport rules described in Biology. These rules correspond to the biological processes in which molecules are transported across membranes in a coupled way, with two molecules passing together across a membrane through a specific protein channel either in the same direction, called *symport*, or in opposite directions, called *antiport*.

These processes were first formalised [32] in membrane computing area in the form of 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 arbitrary size. In the first case, all objects of multiset x pass together across the membrane with which the rule is associated, in the second case the objects of z are sent out and those of w are brought into the membrane. The length of x, denoted by $|x|$, is called the *weight* of a symport rule as above, and max($|z|$, $|w|$) is the *weight* of the antiport rule.

Such rules can be used in a *symport/antiport membrane system*, which 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 labelled 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 membranes of μ,
6. $i_o \in \{1, 2, 3, \ldots, m\}$ is the label of a membrane of μ, which indicates the *output* region of the system.

The rules in the R_i's can be of two types, symport rules and antiport rules, of the forms as specified above.

Again, the rules are used in the nondeterministic 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 computed in this way by Π is denoted by $N(\Pi)$.

There are also other variants considered in literature, both in what concerns the form of rules or combinations of the types of rules, and in what concerns the way of controlling the use of the rules, but we do not continue here in this direction, delaying it for Section 5.3.

4. Symport/antiport systems results

A large number of papers have been written concerning the model described in the previous section, symport/antiport systems. We will use this model to touch upon the topic dealing with the computational universality (i.e. equivalence with Turing machines) of such systems.

Many results about the original (nondeterministic) variant of these systems have been reported. The restrictions are on the number of membranes, size of rules or number of objects used. When considering the first two, universality can be obtained with 3 membranes and symport of size 2 (i.e. at most 2 objects are involved in any symport rule) or with 2 membranes and symport of size 3. If one considers both symport and antiport rules, then the results show that universality is reached with one membrane and uniport (symport of size 1) and antiport of size 2. If minimal symport/antiport are considered, then the universality is reached with three membranes. For all these results we refer the interested reader to Ref. [2]. We note that the majority of the universality proofs are obtained by simulating with the membrane system the work of a counter automaton or its close 'relative', register machine.

Last year several results considering the size of the alphabet were also reported. It is important to note that in this setting the authors try to limit two of the three features mentioned above, the number of objects in the alphabet and the number of membranes with a trade-off in the size of the rules used. The following combinations were proved universal: 2 objects and 4 membranes, 3 objects and 3 membranes, 4 objects and 2 membranes, 5 objects and 1 membrane. Symport/antiport systems with s objects and m membranes are universal if $s + m = 6$ and $s \neq 1$, $m \geq 1$. We do not know at this time whether these results are optimal.

4.1 Generators vs. acceptors

The results mentioned above were obtained for the original variant of symport/antiport systems, the one that works in a generative manner, the system starts in the initial configuration, and through a succession of steps accumulates the result encoded in multiplicities of objects in a particular membrane. We now consider a different setting: the machine is given a value as input which is encoded as a multiplicity of a special object, and has to decide whether it accepts the input value or not. In this way we can again define in the set of numbers computed/accepted by such a device. In such a setting the machine can be either working in a nondeterministic manner, like the generator version of symport/antiport systems, or in a deterministic manner.

Let us now look at deterministic symport/antiport system acceptors with 1 or 2 membranes and 1–3 objects. In this setting the results are stronger than the nondeterministic generative case suggesting that the acceptors are more 'powerful'.

Let G be an m-membrane S/A system with alphabet V and $E \subseteq V$ be the set of objects that occur abundantly in the environment. Each object in VE has a bounded number of copies in the environment. If $|E| = k$, then we say the system is an m-membrane k-symbol symport/antiport system.

Let o be a distinguished symbol in E. There is a fixed $w \in (V - E)^*$ such that at the start of the computation, a multiset wo^n for some nonnegative integer n, is placed in the skin membrane. We say that o^n is accepted if the system halts. A deterministic m-membrane k-symbol symport/antiport system is defined as described before. In this setting we have the following:

THEOREM 1. Let $L \subseteq o^*$.

1. L is accepted by a deterministic symport/antiport system using only one membrane and one symbol if and only if it is semilinear.
2. Let $(m, k) \in \{(1,3),(3,1),(2,2)\}$. Then L is accepted by a deterministic m-membrane k-symbol symport/antiport system if and only if it is recursively enumerable.
3. There are recursive sets that cannot be accepted by deterministic 1-membrane 2-symbol symport/antiport systems and deterministic 2-membrane 1-symbol symport/antiport systems.

One can show that Theorem 1 part 1 holds for the nondeterministic case. Obviously, part 2 holds for the nondeterministic version as well. We believe that part 3 also holds for the nondeterministic case, but we have no proof at this time. We now pass to some nonuniversality results reported recently for some particular restrictions for the systems.

4.2 Restricted symport/antiport systems: nonuniversality results

In Ref. [18] some restricted versions of symport/antiport systems were studied. One model, called *bounded symport/antiport system*, has only one membrane and has rules of the form (u, out; v, in) with the restriction that $|u| \geq |v|$. The environment has an infinite supply of every object in V. An input $z = a_1^{n_1} \ldots a_k^{n_k}$ (each n_i a non-negative integer) is accepted if the system when started with wz, where w is a fixed string not containing a_i ($1 \leq i \leq k$) eventually halts. The following result was reported in Ref. [18]:

THEOREM 2. Let $L \subseteq a_1^* \ldots a_k^*$. Then the following statements are equivalent:

1. L is accepted by a bounded symport/antiport system.
2. L is accepted by a log n space-bounded Turing machine.
3. L is accepted by a two-way multihead finite automaton.

This result holds for both deterministic and nondeterministic versions.

The next result follows from Theorem 2 and the following result
Deterministic and nondeterministic two-way multihead finite automata over a unary input alphabet are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary input alphabet) are equivalent.

THEOREM 3. The deterministic and nondeterministic bounded symport/antiport systems over a unary input alphabet are equivalent if and only if the deterministic and nondeterministic linear-bounded automata (over an arbitrary input alphabet) are equivalent. The latter problem is a long-standing open question in complexity theory

We can also consider multi-membrane symport/antiport systems, called special symport/antiport systems, which are restricted in that only rules of the form (u, out; v, in), where $|u| \geq |v|$, can appear in the skin membrane. There are no restrictions on the rules in the other membranes. Thus, the number of objects in the system during the computation cannot increase. The environment does not contain any symbol initially. Only symbols exported from the skin membrane to the environment can be retrieved from the environment. Note that in the bounded symport/antiport system, the environment has an infinite supply of every object in V.

THEOREM 4. Let $L \subseteq a_1^* \dots a_k^*$. Then the following statements are equivalent:

1. L is accepted by a special symport/antiport system.
2. L is accepted by a bounded symport/antiport system.
3. L is accepted by a $\log n$ space-bounded Turing machine.
4. L is accepted by a two-way multihead finite automaton.

This result holds for both deterministic and nondeterministic versions.

Various models of P systems have been investigated and have been shown to be universal, i.e. Turing machine complete, even with a very small number of membranes (e.g. 1 or 2 membranes). The question of whether there exists a model of P systems where the number of membranes induces an infinite hierarchy in its computational power had been open since the beginning of membrane computing. Clearly, for models that are universal, there cannot be a hierarchy. So the hierarchy question makes sense only for non-universal systems. This question was answered in the affirmative in Ref. [18], where it was proved the following result:

THEOREM 5. For every r, there exist an $s > r$ and a unary language L (i.e. subset of o^*) accepted by an s-membrane special S/A system that cannot be accepted by any r-membrane special S/A system. The result holds for both deterministic and nondeterministic versions.

Similary, the number of symbols in the alphabet V of a bounded S/A system induces an infinite hierarchy

THEOREM 6. For every r, there exist an $s > r$ and a unary language L accepted by a bounded S/A system with an alphabet of s symbols that cannot be accepted by any bounded S/A system with an alphabet of r symbols. This result holds for both deterministic and nondeterministic versions.

4.3 Language acceptors and characterization theorems

In this subsection, we give 'syntactic' characterizations of context-sensitive languages (CSLs) and other language classes in terms of certain models of symport/antiport membrane systems. These are the first such characterizations of CSLs in terms of membrane systems. Variations of our model yield characterizations of regular languages, languages accepted by one-way $\log n$ space-bounded Turing machines, and recursively enumerable languages. We note that previous characterizations of formal languages in the membrane computing literature are mostly for the Parikh images. Unlike the models discussed in the previous subsections where we were interested in multisets of (tuples of) numbers, here we use the systems as language (i.e. pattern) acceptors. Thus, the order in which the input symbols are processed is important.

4.3.1 Characterizing the context-sensitive languages

Let us define an exponential symport/antiport acceptor (ESAA) as a 1-membrane SAA $\Pi = (V, \Sigma, [_1]_1, w_1, V - \Sigma, R_1)$ with the components as specified in the previous section, but with the following specific properties: the input alphabet $\Sigma \subseteq V$ contains a distinguished symbol \$ (the end marker), the environment contains all objects from $V - \Sigma$ (and no object from Σ), and the rules are of the following four types:

1. $(u, \text{out}; v, \text{in})$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$.
2. $(u, \text{out}; va, \text{in})$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$, and $a \in \Sigma$. A rule of this type is called a read-rule.
3. $(u, \text{out}; v, \text{in})|_a$, where $u, v \in (V - \Sigma)^*$, and $a \in \Sigma$ (a is a promoter). Note that there is no restriction on the relative lengths of u and v. In particular, the length of v can be greater than the length of u.
4. For every $a \in \Sigma$, there is at least one rule of the form $(a, \text{out}; v, \text{in})$ in the set R_1, where $v \in (V - \Sigma)^*$. Moreover, this is the only type of rules for which a can appear on the left part of the rule.

We now describe how an ESAA accepts a language. Note some important differences from the way a general SAA works.

An input string $x = a_1 \ldots a_n\$$, where a_i is in $\Sigma - \{\$\}$ for $1 \leq i \leq n$, is provided online in the environment. The symbols are brought into the system in the order they appear in x by means of read-rules, i.e. rules of the form $(u, \text{out}; va, \text{in})$. Maximal parallelism in the application of the rules is assumed as usual. Hence, in general, the size of the multiset of rules applicable at each step is unbounded, and, in particular, the number of instances of read-rules applicable in a step is unbounded. However, the number of read-rules in an applicable multiset cannot exceed the number of symbols of x remaining to be read, and the symbols in these read-rules, say there are s of them, must be consistent with the next s symbols of the input string x that have not yet been processed. Note that rules of types 1, 3 and 4 do not consume any input symbol from x. Note also that because of maximal parallelism, any symbol $a \in \Sigma$ that is imported from the environment by a rule of type 2 is always transported back to the environment in the following step by a rule of type 4. When a rule of type 4 is applied, the symbol a that is exported to the environment does not get inserted to the input string, that is, it is never brought again in the system.

The input string $x = a_1 \ldots a_n\$$ is accepted if, after reading all the input symbols, Π eventually halts.

Note the important fact, essentially used in some proofs, that if the computation halts before reading the end marker \$, then the computation aborts, and no result is provided.

The language accepted is $L(\Pi) = \{a_1 \ldots a_n | a_1 \ldots a_n\$ \text{ is accepted by } \Pi\}$. Note that we do not include the end marker.

We have two versions of ESAAs: nondeterministic and deterministic, where in the deterministic case, the maximally parallel multiset of rules applicable at each step of the computation is unique.

A nondeterministic linear-bounded automaton (NLBA) M is a generalization of a nondeterministic two-way finite automaton with input markers in that the input head can rewrite the symbols on the tape, except the end markers. An input $\$a_1 \ldots a_n\$$ is accepted if M, when started in its initial state on the left end marker, eventually enters an accepting state. The language accepted by M is $L(M) = \{a_1 \ldots a_n | \$a_1 \ldots a_n\$ \text{ is accepted}\}$. It is well-known that an NLBA is equivalent to a linear space-bounded nondeterministic Turing machine. A deterministic linear-bounded automaton will be denoted by DLBA.

NLBAs (DLBAs) accept exactly the context-sensitive (deterministic context-sensitive) languages.

THEOREM 7. A language L is accepted by a nondeterministic (deterministic) ESAA if and only if L is accepted by an NLBA (DLBA).

The next result follows from Theorem 7 and the fact that it holds for DLBAs and NLBAs. Closure under complementation for NLBAs was shown

THEOREM 8.

1. The family of languages accepted by deterministic (nondeterministic) ESAA is closed under union, intersection, complementation, concatenation and Kleene$*$.
2. The membership problem is decidable for nondeterministic ESAAs.
3. The emptiness problem is undecidable for deterministic ESAAs.

Consider now the case of the input alphabet $\Sigma = \{1, 2, \$\}$, i.e. binary, since we ignore the end marker. We also restrict the type 3 rule so that it has the following form:

$$(u, \text{out}; v, \text{in})|_a \quad \text{with} \quad 2|u| \geq |v|.$$

Call the above a 2-ESAA. Then following results were also shown in Ref. [17]:

THEOREM 9. A language $L \subseteq \{1, 2\}^*$ is accepted by an NLBA (DLBA) if and only if it is accepted by a nondeterministic (deterministic) 2-ESAA.

THEOREM 10. (Hierarchy of 2-ESSAs with Respect to the Alphabet size) For every r, there is an $s > r$ such that nondeterministic (deterministic) 2-ESAAs with an alphabet of s symbols can accept more languages than nondeterministic (deterministic) 2-ESAAs with an alphabet of r symbols.

4.3.2 *Variations of the ESAA model*

ESSAs have four types of rules. In this section, we look systematically, considering all possible cases, at the classes of languages generated when one or two types of rules are not allowed.

Let us start by observing that we have two kinds of restrictions in the rules of an ESAA, those related to the length of the strings (in rules of types 1 and 2), and those related to the role of terminal symbols (rules of types 3 and 4). In particular, the obligatory existence of rules of type 4 is a very strong restriction, because we can make use of rules of type 3 (promoted, but without any restriction on the length of the strings) only one step after bringing a terminal into the system. Thus, presumably, removing rules of type 4 add power – and this will be confirmed below.

4.4 Type 2

Only rules of type 2 can bring terminals into the system, hence such rules cannot be avoided. Still, such rules are sufficient to accept at least (as we will see in the next section, *exactly*) all regular languages.

THEOREM 11. Any regular language $L \subseteq T^*$ can be accepted by a deterministic ESAA having only rules of type 2.

REMARK. We observe that the result above is still valid if the input string to Π does not have the end marker \$. One can just modify R_1 to be: $R_1 = \{(s, \text{out}; as', \text{in}) | \delta(s, a) = s'\} \cup \{(s, \text{out}; a, \text{in}) | \delta(s, a) \in F\}$. We note that the same observation about the results being valid when the input strings have no end marker will be true for some other models discussed in this paper. We discuss this issue further later.

4.5 Types 1, 2

Let us now add rules of type 1. Thus, the only rules are of the forms $(u, \text{out}; v, \text{in})$ and $(u, \text{out}; va, \text{in})$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$ and $a \in \Sigma$. Call this new model a regular SAA.

Clearly, for such systems, the input symbols that are read-in during the computation remain in the membrane and not exported back to the environment, so there is no need to keep track of their multiplicities. Moreover, because $|u| \geq |v|$ in rules of type 1, the number of symbols in the membrane does not grow during the computation. Hence, such a system can be simulated by an NFA. Conversely, from Theorem 11 we know that rules of type 2 suffice in order to accept all regular languages, hence we have the following results:

THEOREM 12. A language L can be accepted by a nondeterministic (deterministic) regular SAA if and only if L can be accepted by an NFA (DFA).

THEOREM 13. A language is regular if and only if it can be recognised by a regular SAA using only rules of type 2.

4.6 Types 1, 2, 4

Let us continue by also adding rules of type 4 (hence no rules of type 3 are allowed). Thus, the rules are of the forms $(u, \text{out}; v, \text{in})$ and $(u, \text{out}; va, \text{in})$, where $u, v \in (V - \Sigma)^*$ with $|u| \geq |v|$ and $a \in \Sigma$. And, as before, for every $a \in \Sigma$, there is at least one rule of the form $(a, \text{out}; v, \text{in})$ in the membrane, where $v \in (V - \Sigma)^*$. This model is called linear SAA, or simply LSAA. It can be shown that this model is equivalent to a restricted form of one-way $\log n$ space-bounded Turing machine.

A nondeterministic one-way Turing machine is restricted $S(n)$ space-bounded if for every accepted input of length n, there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by $S(d)$ where $d \leq n$ and d is the number of input tape cells already read, that is, the distance of the reading head from the left end of the one-way input tape [8]. It is interesting the case when $S(n)$ is logarithmic.

THEOREM 14. A language L is accepted by a nondeterministic (deterministic) LSAA Π if and only if L is accepted by a nondeterministic (deterministic) restricted one-way $\log n$ space-bounded Turing machine M.

THEOREM 15. Special SAAs and LSAAs are equivalent. This holds for both nondeterministic and deterministic cases.

THEOREM 16. Deterministic LSAAs are strictly weaker than nondeterministic LSAAs.

Characterizations were also shown in Ref. [17] for the following: Types 2, 4; 2, 3, 4; 2, 3; 1, 2, 3; etc.

4.6.1 *Multi-membrane ESAAs*

One can generalise the ESAAs to have multiple membranes. In the skin membrane, the rules are the four types used for ESSA. The rules in the other membranes can be of the forms: (u, out), (v, in) and (u, out; v, in), where $u, v \in (V - \Sigma)^*$, but there is no restriction on the relative lengths of u and v. Call this generalised model MESAA. Obviously, a nondeterministic (deterministic) MESSA can simulate an NLBA (DLBA). The converse is also easy to see.

Now define a 2-MESAA as in a 2-ESAA. Thus, a 2-MESSA has input alphabet $\{1, 2, \$\}$, and in the skin membrane the rules of type 3 satisfy $2|u| \geq |v|$. Then, from Theorem 9, a nondeterministic (deterministic) 2-MESSA is equivalent to an NLBA (DLBA) over a binary alphabet. Also, as in Theorem 10, fixing the number of membranes to $m \geq 1$, will induce an infinite hierarchy with respect to the number of symbols. Thus, we have:

THEOREM 17. Let m be any integer ≥ 1. Then L is a binary CSL if and only if it can be accepted by a 2-MESAA with m membranes and multiple number of symbols. Moreover, holding the number of membranes at m, there is an infinite hierarchy in computational power (within the class of binary CSLs) with respect to the number of symbols.

Let us now suppose we fix the alphabet of the 2-MESSA to a given size s; then, the question is whether by allowing the 2-MESSA to have multiple membranes, it can simulate any LBA over a binary alphabet.

THEOREM 18. A language $L \subseteq \{1, 2\}^*$ is accepted by an NLBA if and only if it can be accepted by a nondeterministic 2-MESSA with an alphabet of size 14.

THEOREM 19. Let s be any integer ≥ 14. Then a binary language is context-sensitive if and only if it can be accepted by a 2-MESAA with s symbols and multiple number of membranes. Moreover, holding the number of symbols at s, there is an infinite hierarchy in computational power with respect to the number of membranes.

Theorems 10 and 19 say that in order for the restricted 2-MESSAs to accept all binary CSLs, at least one parameter, either the number of symbols or the number of membranes, must grow. As far as we know, these are the first results of their kind in the membrane systems area. They contrast the result from Ref. [1] that (unrestricted) symport/antiport systems with $s \geq 2$ symbols and $m \geq 1$ membranes accept (or generate) exactly the recursively enumerable sets of numbers even for $s + m = 6$.

4.6.2 *Another variant*

The model of ESAA we have studied, including its variants and generalizations, had an online input, $x = a_1 \ldots a_n\$$, and only these symbols (in the order given) can be read into the membrane. The model was defined with the input given this way because it allowed the study of both the deterministic and nondeterministic versions.

We now look at a new variant of the model of a nondeterministic ESAA. The system, which we call ESAA $+$, has alphabet V containing the input alphabet Σ, but we no longer assume that Σ contains an end marker symbol. It has the same four types of rules as in an ESAA, but it accepts a language in the following way. We start from the initial configuration and, as usual, we use the rules in the nondeterministic maximally parallel manner. During the computation, symbols from Σ can be brought into the system, from the environment; these symbols are arranged in a sequence, corresponding to the moments when they enter the system (if several objects come at the same time, then any permutation of them is considered), and, if the computation halts, then the strings defined in this way are said to be accepted.

THEOREM 20. A language L is accepted by a nondeterministic ESAA $+$ if and only if L is accepted by an NLBA.

Finally, we remark that the characterizations for the nondeterministic versions of some of the variants of ESAA we gave earlier remain valid when the input and acceptance are defined as in ESAA $+$.

An interesting open problem is to find similar syntactic characterizations (that is, based on restrictions on the forms of rules of the membrane systems used) for other families of languages, in particular, for context-free languages. Another research topic is find characterizations of known families of languages by using other types of membrane systems. For instance, it is known that symport/antiport systems are universal also when using only symport rules. And as a last question in this section: it is interesting to know whether some of the results we described above hold for such restricted (symport only, etc.) systems.

5. Membrane systems with non-maximal parallelism

As we have seen above, the maximal parallelism is rather useful from a theoretical point of view, both in what concerns the computing power and the efficiency of membrane systems, but it is in some cases not biologically grounded. The discussion is not trivial: is there a unique clock in a cell, marking uniformly the time of each compartment? Otherwise stated, is the cell a synchronised system or an asynchronous one? Some people answer affirmatively, other people answer negatively this question. What seems to be clear is that there is a high degree of parallelism in a cell. When several reactions can take place in the same compartment, many of them actually take place, in parallel. If we wait long enough, then all possible reactions happen. Well, 'waiting long enough' means changing the time unit, hence we return again to the question whether the time is uniform in the whole cell.

In what follows we avoid this discussion and other related issues, and we confine ourselves to a more technical discussion of the versions of parallelism considered so far in membrane computing. We do not present details concerning the sequential systems that were investigated in several places,

5.1 Bounded parallelism

We consider the following new definition of 'maximal parallelism' in the application of evolution rules in a membrane system Π: Let $R = \{r_1, \ldots r_k\}$ be the set of (distinct) rules in the system. Π operates in maximally parallel mode if at each step of the computation, a maximal subset of R is applied, and at most one instance of any rule is used at every step (thus at most k rules are applicable at any step). We refer to this system as a maximally parallel system. We can define three semantics of parallelism. For a positive integer $n \leq k$, define:

n-Max-Parallel: At each step, nondeterministically select a maximal subset of at most n rules in R to apply. This implies that no larger subset is applicable.

$\leq$ *n-Parallel*: At each step, nondeterministically select any subset of at most n rules in R to apply.

n-Parallel: At each step, nondeterministically select any subset of exactly n rules in R to apply.

In all three cases, if any rule in the subset selected is not applicable, then the whole subset is not applicable. When $n = 1$, the three semantics reduce to the *Sequential* mode.

The three modes of parallelisms were studied for two popular models membrane systems: multi-membrane catalytic systems and communicating membrane systems. It was shown that for these systems, *n-Max-Parallel* mode is strictly more powerful than any of the following three modes: *Sequential*, $\leq$*n-Parallel*, or *n-Parallel*. For example, it follows from the result that a maximally parallel communicating membrane system is universal for $n = 2$. However, under the three limited modes of parallelism, the system is equivalent to a vector addition system, which is known to only define a recursive set. Some results are rather surprising. For example, it was shown that a *Sequential* 1-membrane communicating membrane system can only generate a semilinear set, whereas with k membranes, it is equivalent to a vector addition system for any $k \geq 2$. Thus the hierarchy collapses at 2 membranes – a rare collapsing result for non-universal membrane systems.

A simple cooperative system (SCO) is a membrane system where the only rules allowed are of the form $a \longrightarrow v$ or of the form $aa \longrightarrow v$, where a is a symbol and v is a (possibly null) string of symbols not containing a. It was also shown that a 9-*Max-Parallel* 1-membrane SCO is universal.

5.2 *Minimal parallelism*

The notion of minimal parallelism looks rather 'bio-realistic' and relaxed: if a compartment of a cell can evolve, then it has to evolve at least in the minimal way, by using at least one reaction. In terms of membrane systems, the condition is the following: we have sets of rules associated with compartments (as in the case of transition membrane systems) or with membranes (as in symport/antiport systems or in membrane systems with active membranes); then, if in a given configuration from such a set at least one rule can be applied, then at least one must be applied. How many rules are actually applied is not prescribed; exactly one rule, more, all rules which can be used, all these possibilities are allowed, we only know that for sure at least one is effectively applied.

It is clear that this way of using the rules is much weaker than the maximal parallelism, because we have a rather limited knowledge about the evolution of the system – the degree of nondeterminism is increased, we cannot get an information about all objects from a compartment, etc. Still, we have one level of parallelism, because the compartments/membranes evolve simultaneously, synchronised, and with a minimal action taking place in each of them.

It is somewhat surprising that this minimal control on the evolution of 'sub-systems' of a membrane system still ensures the computational universality. This was proved

for symport/antiport membrane systems working in the generating or the accepting mode (in the second case also for deterministic systems), and for membrane systems with active membranes. In all these cases the results was obtained with some of the parameters (especially, the number of membranes) being larger, thus compensating for the loss of control when passing from maximal parallelism to minimal parallelism.

The ‘explanation’ of the previously mentioned results lies in the way the proof is conducted: because we have to simulate a sequential computing device (a register machine), the parallelism is low anyway, hence the construction can be arranged in such a way that either exactly one rule is applicable for each membrane, or only a small number of rules can be applied, hence for the constructed systems minimal parallelism is very close (if not identical) to maximal parallelism.

It was shown in Ref. [6] that efficiency results can also be obtained in the case of the minimal parallelism like in the case of maximal parallelism: the satisfiability of any propositional formula in the conjunctive normal form, using n variables, can be decided in linear time with respect to n by a membrane system with active membranes, using rules of types (a)–(c) and (e), while working in the minimally parallel mode.

Results as above, both universality and efficiency, were recently reported in Ref. [25] for other types of membrane systems, based on operations similar to membrane division, such as membrane separation: rules $[_hQ]_h \rightarrow [_hK]_h[_hQ - K]_h$, with the meaning that membrane h is split into two membranes, one containing the objects from the set K and the other one containing the complement of K with respect to Q.

5.3 Systems with proteins on membranes

We have mentioned in the Introduction that part of the biochemistry taking place in a cell is controlled by proteins (enzymes) bound on membranes. In particular, the symport/antiport processes depends on the protein channels embedded in the membranes. Thus, it is natural to also consider objects (proteins) placed on membranes, not only in the compartments of a membrane system. This is the approach of so-called brane calculi introduced in Ref. [3], which deals only with operations with membranes controlled by their proteins. Perhaps more realistic is to combine the two approaches, thus considering objects both in the compartments of a membrane structure and on the membranes. There are already several papers dealing with such a bridge between membrane computing and brane calculi; we refer for references.

An idea relevant for our discussion of the parallelism in membrane system is that considered : let us consider objects in compartments and proteins on membranes; the objects can pass across membranes like in symport/antiport systems and can also evolve like in transition membrane systems under the control of proteins placed on membranes. In turn, these proteins can change, either without restrictions, or only in a flip–flop manner like bi-stable catalysts. The most general form of rules in this framework is the following one:

$$a[_ip|b \rightarrow c[_ip'|d,$$

where a, b, c, d are objects and p, p' are proteins; the notation $[_ip|$ means ‘membrane i has protein p placed on it’. When applying this rule, the object a placed outside membrane i and object b placed inside membrane i are simultaneously changed in c, d, respectively, assisted by protein p, placed on the membrane, which evolves to protein p'.

A membrane system using rules of this form starts with multisets of objects placed in the compartments, in the environment, and on membranes called proteins. Because the proteins cannot leave the membranes and cannot multiply, their number is constant during the computation – hence finite. This means that the parallelism of the system is also bounded; in each step at most as many rules can be used as the number of proteins appearing on the membranes. Thus, the protein behaves like catalysts for symport/anti-port-like rules, where the objects may also evolve.

The generality, hence the power of the rules in the form given above is clear, so a series of restrictions were considered ; the protein should not evolve, or it should only oscillate between two states, p, p'; the objects cannot change, but only move across the membrane; only one of a and b is present, and it either evolves and remains in the same region, or moves to another region without evolving, or both evolves and moves to another region. A large number of types of rules are obtained, and a large number of combinations of such rules can then be considered and investigated in what concerns the computing power. In most cases, universality results were again obtained. Details can be found in

6. Catalytic systems

There are many other variants which deserve to be considered in relation with membrane computing, but due to space restrictions were not covered. We will try to give here brief descriptions for another one of these variants, the catalytic systems.

A catalytic system has rules of the forms: $Ca \longrightarrow Cv$ or $a \longrightarrow v$, where C is a catalyst, a is a noncatalyst symbol, and v is a (possibly null) string of noncatalyst symbols. A catalytic system whose rules are only of the form $Ca \longrightarrow Cv$ is called purely catalytic system.

Such systems can work in a generative or accepting mode as well as using the rules either in a deterministic or nondeterministic fashion.

The catalytic systems working in a nondeterministic way were shown to be universal for one membrane and two catalysts, but these proofs of universality involve a high degree of parallelism and they could not be extended to the deterministic case. In fact it was shown using a graph-theoretic approach that the Parikh map of the language $\subseteq a_1^* \ldots a_k^*$ accepted by any deterministic catalytic system is a simple semilinear set which can also be effectively constructed. This result gives the first example of a membrane system for which the nondeterministic version is universal, but the deterministic version is not. For deterministic 1-membrane catalytic systems using only rules of type $Ca \longrightarrow Cv$, it was shown that the set of reachable configurations from a given initial configuration is effectively semilinear. In contrast, the reachability set is no longer semilinear in general if rules of type $a \longrightarrow v$ are also used. This result generalises to multi-membrane catalytic systems.

Deterministic catalytic systems which allow rules to be prioritised were also considered Three such systems, namely, *totally prioritised, strongly prioritised* and *weakly prioritised* catalytic systems, were investigated in the literature. For totally prioritised systems, rules are divided into different priority groups, and if a rule in a higher priority group is applicable, then no rules from a lower priority group can be used. For both strongly prioritised and weakly prioritised systems, the underlying priority relation is a *strict partial order* (i.e. irreflexive, asymmetric and transitive). Under the semantics of strong priority, if a rule with higher priority is used, then no rule of a lower priority can be used even if the two rules do not compete for objects. For weakly prioritised systems, a rule is applicable if it cannot be replaced by a higher priority one. For these three prioritised systems, the results were contrasting: deterministic strongly and weakly prioritised catalytic systems are universal, whereas totally prioritised systems only accept semilinear sets.

7. Final remarks

We have touched on a series of issues related to membrane systems, as final remarks we mention the fact that a series of applications of these systems were reported in the last years, especially in modelling and simulating processes in the cell, but also related to populations of cells (e.g. of bacteria). The attractiveness of membrane computing from this point of view comes from several directions: membrane systems are directly inspired from biology, are easy to understand by biologists, easy to extend (scale-up), modify, program, are nonlinear in behaviour, and suitable for situations where we deal with small populations of molecules and/or with slow reactions. In such cases the traditional models based on differential equations are not adequate. Because of this, variants that consider the law of mass action in the rule application were used in to model the cascade of reactions in signalling pathways for EGFR and in to model FAS induced apoptosis. This discrete technique provided quite interesting results when compared to the differential simulations.

There are many other interesting topics to be discussed in this area, we mention briefly the variants that model the membrane division. In that setting interesting results have been obtained by a trade-off between time and space. Effectively by using the membrane division it was shown that one can solve combinatorially intractable problems, e.g. NP-complete and PSPACE-complete in polynomial time (in some cases even linear time). Of course these results refer to the nondeterministic maximally parallel devices discussed, thus the famous open problem related to the equality between P and NP is not settled.

It is then no surprise to notice the increasing number of simulations/implementations of various classes of membrane systems. There are also proposals for implementing membrane systems on a dedicated, reconfigurable, hardware, as done or on a local network, as reported

Acknowledgements

The research of O. H. Ibarra was supported in part by NSF Grants CCF-0430945 and CCF-0524136. The research of A. Păun was supported in part by LA BoR RSC grant LEQSF (2004–07)-RD-A-23 and NSF Grants IMR-0414903 and CCF-0523572.

References

A. Alhazov, R. Freund, and M. Oswald, *Symbol/membrane complexity of P systems with symport/antiport rules*, in *Pre-Proceedings of the 6th International Workshop on Membrane Computing, WMC6*, R. Freund, G. Lojka, M. Oswald and Gh. Păun, eds., Vienna Technological University, Vienna, 2005, pp. 123–146.

A. Alhazov, R. Freund, and Y. Rogozhin, *Computational power of symport/antiport: history, advances and open problems*, in *Membrane Computing, International Workshop, WMC6, Vienna, Austria, 2005, Selected and Invited Papers*, Lecture Notes in Computer Science 3850, R. Freund, Gh. Păun, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 2006, pp. 1–30.

L. Cardelli, *Brane calculi. Interactions of biological membranes*, in *Computational Methods in Systems Biology. International Conference CMSB 2004, Paris, France, May 2004, Revised Selected Papers*, Lecture Notes in Computer Science 3082, V. Danos and V. Schachter, eds., Springer-Verlag, Berlin, 2005, pp. 257–280.

S. Cheruku et al., *Simulating FAS-induced apoptosis by using P systems*, Prog. Nat. Sci. 17(4) (2007), pp. 424–431.

G. Ciobanu and G. Wenyuan, *A P system running on a cluster of computers*, in *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers*. Lecture Notes in Computer Science, 2933, C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 2004, pp. 123–139.

G. Ciobanu et al., *P systems with minimal parallelism*, Theor. Comput. Sci. 378(1) (2007), pp. 117–130.

G. Ciobanu, Gh. Păun and M. J. Pérez-Jiménez, eds., in *Applications of Membrane Computing*, Springer, Berlin, 2006.

E. Csuhaj-Varju, O. H. Ibarra, and G. Vaszil, *On the computational complexity of P automata*, in *Proceedings of DNA10 Conference, Milano, 2004*, Lecture Notes in Computer Science, 3384, C. Ferretti, G. Mauri and C. Zandron, eds., Springer-Verlag, Berlin, 2005, pp. 76–89.

Z. Dang, and O. H. Ibarra, *On P systems operating in sequential and limited parallel modes*, in *Pre-Proceedings of the Workshop on Descriptional Complexity of Formal Systems, DCFS*, University of Western Ontario, London, 2004, pp. 164–177.

R. Freund, *Asynchronous P systems and P systems working in the sequential mode*, in *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revised Papers*, Lecture Notes in Computer Science, 3365, G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 2005, pp. 36–62.

R. Freund, and A. Păun, *Membrane systems with symport/antiport rules: universality results*, in *Proceedings of the WMC-CdeA2002*, Lecture Notes in Computer Science, 2597, Springer-Verlag, Berlin, 2003, pp. 270–287.

R. Freund et al., *Computationally universal P systems without priorities: two catalysts suffice*, Theor. Comput. Sci. 330(2) (2005), pp. 251–266.

R. Freund, et al. eds., *Pre-Proceedings of the 6th International Workshop on Membrane Computing, WMC6*, Vienna Technological University, Vienna, 2005.

R. Freund, et al. eds., *Membrane Computing, International Workshop, WMC6, Vienna, Austria, 2005, Selected and Invited Papers*, Lecture Notes in Computer Science, 3850, Springer-Verlag, Berlin, 2006.

Cellular Computing. Complexity Aspects., in M. A. Gutiérrez-Naranjo, Gh. Păun & M. J. Pérez-Jiménez eds., Fenix Editora, Sevilla, 2005.

O. H. Ibarra, *Some computational issues in membrane computing*, in *Proceedings of the 30th International Symposium, MFCS 2005*, Lecture Notes in Computer Science, 3618, Springer-Verlag, Berlin, 2005, pp. 39–51.

O. H. Ibarra, and G. Păun, *Characterizations of context-sensitive languages and other language classes in terms of symport/antiport P systems*, Theor. Comput. Sci. 358 (2006), pp. 88–103.

O. Ibarra, and S. Woodworth, *On bounded symport/antiport systems*, in *Pre-Proceedings of 11th International Meeting on DNA Computing*. Lecture Notes in Computer Science, 3850, 2006, pp. 49–54.

O. H. Ibarra, and H. C. Yen, *On deterministic catalytic P systems*, in *Proceedings of the CIAA05*, Lecture Notes in Computer Science, 3845, Springer-Verlag, Berlin, 2006, pp. 164–176.

O. H. Ibarra, H.-C. Yen, and Z. Dang, *On various notions of parallelism in P systems*, Int. J. Found. Comput. Sci. 16(4) (2005), pp. 683–705.

O. H. Ibarra et al., *Normal forms for spiking neural P systems*, Theor. Comput. Sci. 372 (2–3) (2007), pp. 196–217.

O. H. Ibarra et al., *On the computational power of 1-deterministic and sequential P systems*, Fund. Inform. 73(1–2) (2006), pp. 133–152.

N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput. 17(5) (1988), pp. 935–938.

M. Ionescu, Gh. Păun, and T. Yokomori, *Spiking neural P systems*, Fund. Inform. 71(2–3) (2006), pp. 279–308.

T.-O. Ishdorj, *Minimal parallelism for polarizationless P systems*, Lecture Notes in Computer Science, 4287, Springer-Verlag, Berlin, 2006, pp. 17–31.

S. N. Krishna, *Combining brane calculus and membrane computing*, Submitted. (2006).

T. Y. Nishida, *An application of P system: a new algorithm for NP-complete optimization problems*, in *Proceedings of the 8th World Multi-Conference on Systems, Cybernetics and Informatics*, N. Callaos, et al. eds.,Vol. V, 2004, pp. 109–112.

Gh. Păun, *Computing with membranes*, J. Comput. Syst. Sci. 61(1) (2000), pp. 108–143, (and Turku Center for Computer Science – TUCS Report 208, November 1998, www.tucs.fi).

Gh. Păun, *P systems with active membranes: attacking NP-complete problems*, J. Automata Lang. Comb. 6(1) (2001), pp. 75–90.

Gh. Păun, *Computing with Membranes: An Introduction*, Springer, Berlin, 2002.

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

A. Păun, and Gh. Păun, *The power of communication: P systems with symport/antiport*, New Generat. Comput. 20(3) (2002), pp. 295–306.

A. Păun, and B. Popa, *P systems with proteins on membranes*, Fund. Inform. 72(4) (2006), pp. 467–483.

A. Păun, and B. Popa, *P systems with proteins on membranes and membrane division*, in *Proc. of the Tenth International Conference on Developments in Language Theory (DLT)*, 26–29 June, Santa Barbara: CA 2006, pp. 292–303.

Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, *Spike trains in spiking neural P systems*, Int. J. Found. Comput. Sci. 17(4) (2006), pp. 975–1002.

M. J. Pérez-Jiménez, and F. J. Romero-Campero, *P systems, a new computational modeling tool for systems biology*, Trans. Comput. Sys. Biol. 4 (2006), 176–197.

B. Petreska, and C. Teuscher, *A hardware membrane system*, in *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers*. Lecture Notes in Computer Science, 2933, C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 2004, pp. 269–285.

A. Rodriguez-Paton and P. Sosik, Membrane computing and complexity theory: Characterization of PSPACE. J. Comput. Sys. Sci. 73(1) (2007), pp. 137–152.

F. J. Romero-Campero and M. J. Pérez-Jiménez, *A study of the robustness of the EGFR signalling cascade using continuous membrane systems*, Lecture Notes in Computer Science, 3561, Springer-Verlag, Berlin, 2005, pp. 268–278.

W. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. Syst. Sci. 4(2) (1970), pp. 177–192.

W. Savitch, *A note on multihead automata and context-sensitive languages*, Acta Inform. 2 (1973), pp. 249–252.

P. Sosik, *P systems versus register machines: two universality proofs*, in *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA Curtea de Arges, Romania*, 2002, pp. 371–382.

A. Syropoulos et al., *A distributed simulation of P systems*, in *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers*. Lecture Notes in Computer Science, 2933, C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 2004, pp. 355–366.

R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform. 26(3) (1988), pp. 279–284.

The Web Page of Membrane Computing, http://psystems.disco.unimib.it.

C. Zandron, C. Ferretti, and G. Mauri, *Solving NP-complete problems using P systems with active membranes*, in *Unconventional Models of Computation*, I. Antoniou, C. S. Calude and M. J. Dinneen, eds., Springer, London, 2000, pp. 289–301.