Further Twenty Six Open Problems in Membrane Computing

Gheorghe Păun

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

Summary. This is a sort of personal list of problems and research topics, compiled with the occasion of the Third Brainstorming Week on Membrane Computing, Sevilla, 2005.

1 Introduction

These notes follow the tradition of [16] and [18] – also combining their titles. . . – having as the simple aim to challenge the reader – if not to address these problems, at least to produce his/her own list of problems and circulate it. Being a (more or less) personal list, the statements are somewhat elliptical and the references pretty scarce – and, of course, the selection is subjective. For technical and/or bibliographical details, one may consult the monograph [17], the recent volume [6], and, especially, the web page from `http://psystems.disco.unimib.it`. A very useful idea is to contact the people who have already worked about/around the problems, in order to get recent information and, the ultimate aim of these notes, to start collaborating[1].

The last warning: the ordering of the problems has no significance (the labelling is only used for an easy reference), while including a problem in the present list does not mean that it is more important/interesting/challenging than any problem not

[1] During the Sevilla Brainstorming several of the problems which follow were addressed and partially solved, or related results were reported. This is the case with problems F, G, K, N, S, W – see the related papers in the present volume or in the companion volume M.A. Gutiérrez–Naranjo, Gh. Păun, M.J. Pérez–Jiménez, eds., *Cellular Computing. Complexity Aspects*, Fenix Editora, Sevilla, 2005. In turn, O.H. Ibarra has recently announced that the results from [11] were extended to P systems with symport/antiport rules, thus solving problem O.

considered here. This does not even mean that the problem is difficult or properly formulated; some of the research topics are mere intuitions and they directly ask for a more precise statement.

2 Problems, Problems, Problems. . .

A Let us start with some problems already "classic" in membrane computing, related to complexity classes, especially concerning the class of problems which can be solved in polynomial time by (recognizing) P systems of a given *type*. Let us denote it by $\mathbf{PMC}_{type}$. Up to now, mainly P systems with membrane division were considered, hence already we have a problem here: consider separately membrane division for elementary membranes (div-e), membrane division allowed also for non-elementary membranes (div-ne), membrane creation (cre), string replication (sre), and compare the four classes $\mathbf{PMC}_{type}$, for *type* $\in \{$*div-e, div-ne, cre, sre*$\}$. How do they compare to each other?

B All the previous classes include **NP**, while from [26] (see also [4]) we know that $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{div-ne}$. Which is the relation of **PSPACE** with the other three classes? It seems that $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{cre}$, [23], too. It is an intriguing question here whether also $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{div-e}$ holds (the conjecture was formulated – P. Sosik, M.J. Pérez–Jiménez – that this is not true; if confirmed, this will be a very interesting result, indeed, showing the need for the division of non-elementary membranes in order to cover **PSPACE**).

C The membrane computing complexity classes are defined in a way similar to the usual way of defining classes **P, NP** in standard complexity theory: one starts from the problem to solve and one constructs a family of P systems solving the instances of the problem (after introducing encodings of the instances in the initial configurations of the respective systems); the construction is done in polynomial time by a Turing machine. This is the so–called *uniform* way to proceed.

A more relaxed possibility is to construct a P system associated with a given instance of the problem – this is called the *semi–uniform* way. This problem was formulated also in other contexts but it deserves to be repeated, maybe it will be sometime settled: do the complexity classes defined in the uniform and the semi–uniform ways differ?

The interest of this question lies in the fact that in molecular computing (see, e.g., the famous Adleman's experiment) one proceeds in the semi–uniform manner – from a practical point of view, it would be a great achievement to have solutions to real life problems, even if dealing with instances of them (the fact that we construct the system solving the problem in a polynomial time makes sure that the problem itself cannot be solved, in a general (algorithmic) way, during this pre–computation time).

D A related and similar problem: the functioning of a P system solving a decision problem can be deterministic, strongly confluent, or weakly confluent. In the last two cases, the system can behave non-deterministically, but the computation must finish in a confluent manner. Strong confluence means that irrespective how the system evolves, at some time it will reach a unique configuration and, after that, the computation continues in a deterministic manner. In the case of the weak confluence the system may evolve non-deterministically the whole computation, but irrespective which is the path followed in the computation tree, the answer the system gives to the problem it has to solve is always the same (we deal with decidability problems, hence either all computations answer YES or all computations answer NO). While most of the polynomial solutions to **NP**–complete problems were given in terms of deterministic computations, still the problem remains to compare the three possibilities. For instance, it would be of interest to find classes of P systems for which systems working in the weakly confluent manner can be simulated by systems working in strongly confluent manner, and, similarly, to find classes of systems for which it is possible to pass from a strongly confluent system to a deterministic one – in all cases, hopefully, with a linear or at most a polynomial slow–down. A positive answer to this problem is again of a practical interest: instead of constructing a restricted type of systems (presumably, this is a more difficult task, as the time complexity of the construction and/or in what concerns the descriptional complexity of the system), it is enough to construct a system of a more relaxed type, and, if needed, we transform it into a system of a more restricted type.

E An "inverse" problem: instead of starting with membrane computing complexity classes defined in a way which imitates the standard classes in complexity theory, and then compare them with Turing complexity classes, try to characterize Turing complexity classes in terms of P systems. For instance, $\mathbf{NP} \subseteq \mathbf{PMC}_{type}$ for many types of P systems – see above. What kind of restrictions should be imposed on the form (type of rules? type of membrane structure? size parameters?) of P systems of a given *type* and/or on the way of using them (bounding the parallelism? bounding/consuming some resources?) in order to get equalities? This reminds of the characterization of the $\mathbf{P} \neq \mathbf{NP}$ conjecture in terms of P systems (see [25] and the references therein) – without having up to now any characterization of the classes **P** and **NP** themselves in this framework.

F Let us consider now some more technical questions, from the same area, of complexity classes. My favorite question (favorite because it is intriguing, even annoying, and related to the very interesting issue of the borderline between efficiency – the possibility of solving Turing hard problems in a feasible membrane computing time – and non-efficiency) is that about the number of "electrical charges", whether or not polarizations are necessary. Initially, P systems with active membranes were introduced with membranes having three possible charges, $+, -, 0$, like in physics. During the previous brainstorming, [3], it was shown that (both for universality and) for solving **NP**–complete problems two charges are enough.

Can the polarizations be completely avoided? (This means to have all membranes, always, of the same polarization, which amounts to say that this parameter will play no role in the system functioning.) The feeling is that this is not possible – and such a result would be rather sound: passing from no polarization (which, in fact, means one polarization) to two polarizations amounts to passing from non-efficiency to efficiency. (This would also raise the question of characterizing the complexity class $\mathbf{PMC}_{div-e1}$, where by *div*-$e1$ we have denoted the family of P systems with active membrane, using the division of elementary membranes, and no polarization.)

G In all previous problems, when dealing with membrane division and membrane creation, one works with cell–like systems, with symbol–objects processed by multiset–rewriting rules – and *never* symport/antiport systems were considered. The possibility of solving `SAT` in polynomial time by means of tissue–like P systems with cell division was shown in [21]; it seems that also cell–like systems with symport/antiport rules can be used in this aim, [2]. However, the use of (cell–like) systems with symport/antiport rules deserves a systematic investigation: using both membrane division and membrane creation, with the case of membrane division considered in the two cases – dividing or not also elementary membranes; comparing the obtained complexity classes with complexity classes based on multiset–rewriting rules; Milano theorems (are membrane division and membrane creation necessary in order to get efficiency in the case of symport/antiport systems?).

H Comparing two complexity classes could mean either proving a strict inclusion or even incomparability, which supposes the (hard) task of finding counterexamples, or separation by means of known complexity classes already separated, or proving the inclusion/equality, and this asks for the simulation of a system of a given type by a system of another type, without losing the polynomial behavior.

This suggests a more general research issue: proving simulation results. Start with a system of a given type A and find an equivalent system of another type, B. If the systems of type A are known to be computationally universal, this will give "for free" the universality of systems of type B. When dealing with solving problems (hence with complexity matters), in order to be of interest, the simulation should have some basic properties: (i) to preserve the determinism/confluency (strong or weak), and (ii) to have at most a polynomial slow–down. A strategy of this type is of a particular interest when introducing a new class of P systems; instead of proving the universality or the efficiency in a direct manner, returning again and again to matrix grammars or register machines, and, respectively, to `SAT` or `HPP`, it would be more interesting to prove that the new systems can simulate systems from an old class, with known properties, thus importing these properties to the new class. A strategy of this type was recently followed in [13].

I Let us now consider a more general question, not very clearly formulated. The "polynomial solutions" to **NP**–complete problems were obtained up to now systematically by making use of a time–space trade–off, based on the (biologically inspired) possibility of creating an exponential workspace in a polynomial (actually, in most cases, linear) time, e.g., by membrane division. However, the membrane computing framework contains many ingredients which, at the first sight, can be used for building "efficient computers". For instance, the maximal parallelism ensures the fact that all objects from a compartment of a P system are simultaneously processed, in one time unit, irrespective how many they are. Similarly, by means of rules with promoters or with inhibitors we can check, again in one time unit, whether a multiset contains or not a given object, irrespective of the size of the multiset. (In some sense, this resemble the use of superposition in quantum computing, when a register of length n contains 2^n data, which are processed simultaneously.) If these ingredients are not sufficient for an essential speed–up, what about considering other possibilities?

Here are a few suggestions. Use rules with promoters/inhibitors which check the presence/absence of the condition objects in the whole system, not only in the region where the rule is applied. In [19] one proposes an operation with membranes opposite to division: remove one copy of a membrane i if there are two copies of this membrane, adjacent, with the same contents, except for one object a in the first copy and an object b in the second copy; after eliminating, say, the first copy, the object b becomes c (that is, the rule is of the form $[_i\, a\,]_i[_i\, b\,]_i \rightarrow [_i\, c\,]_i$, applied only if the two copies of the membrane i have the same contents, excepting the objects a, b). Any usefulness of this powerful operation? Then, in [12] one proposes communication indications of the forms in^* and out^*, with the meaning that an object a having associated the target indication in^* will immediately go to one of the elementary membranes internal to the region where (a, in^*) was introduced, while (a, out^*) means that a will immediately go to the skin region – in all cases, irrespective which is the distance from the starting region to the target region. Can this kind of "teleportation" be useful for speeding–up computations?

If none of these ideas works, what else can be imagined (not directly leading to a time–space trade–off)?

J In the end of this series of problems dealing with complexity matters, let us recall the important recent proposal from [14] (see also [15]), of using membrane computing ideas in devising approximate algorithms for addressing optimization hard problems. The strategy is similar to that from distributed evolutionary computing (the family of candidate solutions is separated in subsets which are processed separately), with a series of novel ingredients specific to cell–like and tissue–like P systems: hierarchical arrangement of compartments, where a small number of candidate solutions are placed (this is essentially different from evolutionary computing, where large populations of candidate solutions are handled), communication among compartments, dynamic membrane structure (in so–called shrink membrane algorithms, the number of membranes decreases during the computation), etc. This strategy was proved in [14], [15] to be rather useful (better in many cases than simulated annealing, e.g., converging very rapidly to good enough solutions, giving good enough solutions also in worst and average cases – hence the method is trustful, we can save trials without essentially losing the quality of the solution, in many cases also giving almost optimal solutions) for the travelling salesman problem.

Two important research areas are open here: (i) address in the same way other optimization problems, (ii) consider other membrane computing ingredients, that is, devise a variety of membrane algorithms and check their efficiency in various contexts (this is very much the same as the strategy of evolutionary computing). As possible further features, we only mention a few ideas: create or dissolve membrane during computation (when the solution does not improve fast enough, whatever that means, create new membranes); allow (bad) solutions to exit the systems, so that a stop condition can be to reach a state when only one solution is still present; use membrane structures which are not linear (the non-determinism in moving solutions to lower membranes can be useful in avoiding local minima); use multisets (of given size) of solutions, with more copies for better solutions (Nishida uses only two solutions in each compartment).

The fact that approximate solutions to hard optimization problems are looked for in many practical applications makes this approach very appealing – not to speak about the encouraging results from [14], [15].

The so–called compound membrane algorithms from the cited papers start with a phase when separate algorithms are performed, producing the candidate solutions for a second phase, when a usual (cell–like) membrane algorithm is performed. What about implementing the first phase on a distributed computer, or on a cluster/net of computers? This seems to significantly save time, making the approach still more attractive practically.

K Recently, [20], it was proved that cell–like symport/antiport P systems using only three objects are computationally complete. Four membranes were used. Thus, there appear here several natural questions: are systems with less than four membranes (and three objects) universal? (Conjecture: no.) Are systems with only two objects universal? (Conjecture: no.) What can we compute with systems using only one object? Can such systems generate non-semi–linear sets of numbers? (Conjecture: no.) The size of rules used in the proof from [20] is not bounded; can a bound be imposed?

L The result from [20] was extended in [10] to tissue–like P systems, and the universality was obtained for only one object, but using 7 cells. For membrane computing, 7 is a big number... Can the universality be obtained with a smaller number of cells? Characterize the families of numbers computed by tissue–like P systems with only one object, using 1, 2, 3, 4, 5, 6 cells. Like in the previous question, can a bound be imposed on the size of symport/antiport rules without losing the universality?

M Extend the results from [20] and [10] to other classes of P systems. In the case of multiset–rewriting systems, the price to pay for bounding the number of objects seems to be the use of cooperating rules. Still the question seems of interest: how many objects we need for universality? Then, what about P systems with active membranes, with communicative rules in the sense of Sosik, with boundary rules in the sense of Bernardini–Manca, with mobile membranes as in [13], etc.

N Staying closer to symport/antiport systems, let us recall the question concerning the power of systems with only two membranes and minimal symport or minimal antiport (three membranes are known to ensure the universality for P systems with with rules of the form $(a, out; b, in), (a, out), (a, in)$ or of the form $(ab, out), (ab, in)$). This is an interesting technical problem, related to the borderline between universality and non-universality. From time to time, rumors are spread about solving this problem in one way or the other, but up to now no proof was circulated and validated.

O There are a series of sound results by O.H. Ibarra dealing with P systems with communicative rules, in the sense of Sosik: $a \rightarrow a_{tar}$, $ab \rightarrow a_{tar_1}b_{tar_2}$, $ab \rightarrow a_{tar_1}b_{tar_2}c_{come}$, with a, b, c objects and $tar, tar_1, tar_1 \in \{here, out, in\}$, with rules of the third type only used in the skin region (such a rule brings an object c from the environment). One of the most interesting results dealing with such systems is that from [11], where a class of P systems is found for which the deterministic systems are strictly less powerful than the non-deterministic systems. Can these results be extended to symport/antiport systems? Note that the question does not have a direct answer, based on simulating communicative rules by means of symport/antiport rules: for instance, a rule $ab \rightarrow a_{out}b_{in}$ does not have a counterpart in terms of antiport rules.

P Concerning the deterministic proofs of universality (of course, for P systems working in the accepting mode, which accept a number introduced in the initial configuration, or for P automata, where we accept the sequence of symbols brought into the system during a computation), this was possible for most types of systems, with a major exception: catalytic systems with multiset–rewriting rules (known to be universal even when using only two catalysts, [9]). Filling in this gap would be a nice result. Proving that deterministic catalytic systems (working in the recognizing mode) are not universal would be a still nicer result.

Q What about universality results for deterministic cell-like systems with three objects or tissue–like systems with only one object? If necessary, increase the number of objects. The same question should be asked for all possible classes of P systems for which problem **M** would be solved.

R One more (possible) technical problem related to symport/antiport systems. Let us relate the idea of catalysts with that of symport/antiport rules and consider certain distinguished objects $C \subseteq O$, such that some rules contain an object from C. If the catalysts are not available in the environment, this can lead to an easy way to inhibit the parallelism. Any usefulness of this idea? (For instance, are P systems with two membranes universal when using minimal catalytic symport/antiport rules – without counting the catalysts in the size of rules)? What about purely catalytic systems, where each rule must contain a catalyst (is this an essential restriction)?

S Let us now start a series of problems – actually, research topics – of a general type, with not–too–precise formulations, with the formulation itself being part of the problem.

First, let us remark the fact that a Turing machine works with a tape which is divided into "cells", but the notion is used in a very restricted and local manner. What about considering that these cells are "real cells", that is, membranes containing multisets of symbols, arranged in a linear manner. In some sense, we have a tissue–like system, but we consider an "infinite tape", hence an infinite string of membranes, and no interaction with the environment.

A possible formal definition would be like the following one: consider a construct (we call it *cell Turing machine*)

$$\Pi = (O, k, w_1, \dots, w_k, w, R),$$

where:

1. O is an alphabet (of objects),
2. $k \geq 1$ is the *degree* of the machine,
3. $w_1, \dots, w_k$ are strings over O, representing the initial contents of cells $1, 2, \dots, k$ of the machine,
4. w is the contents of cells $k+1, k+2, \dots$ (the same for all cells from cell $k+1$ to infinity),
5. R is a finite set of rules of the following two types:
   (a) $(i, x; y, i+1)$, for $1 \leq i \leq k$, $x, y \in O^*$ (usual antiport rules),
   (b) $(*, x; y, *+1)$, for $x, y \in O^*$ (antiport rules without labels of cells).

The intuition is that the tape of the machine has the first k cells (the tape is finite to the left and infinite to the right) "differentiated", with different contents and different rules, and all other cells are "non-differentiated", they have the same contents and rules which can be applied irrespective which is the cell (that is, any rule $(*, x; y, *+1)$ can be applied for exchanging the multisets x, y among any two neighboring cells $j, j+1$, with $j > k$).

The machine recognizes a string $w = a_1 a_2 \dots a_n \in O^+$ of length n as follows: we introduce a_i in cell i, for $1 \leq i \leq n$ (irrespective which is the relation between k and n) – possibly also a marker \$ in cell $n+1$ (that is, these symbols are added to the corresponding multisets w_i, w, respectively); the rules are used in the non-deterministic maximally parallel manner; the string is recognized if and only if the computation halts.

The technical details concerning a more precise definition of the computation in such a device are left to the reader. Maybe also the structure of the machine should be changed.

Now, the list of questions to address in this framework is very large. Which is the power of such a machine? Do the parameter k induces an infinite hierarchy? What about the size of "axioms" $w_1, \dots, w_k$ and w? (Note that, because we do not communicate with the environment, we need the symbols from the non-differentiated cells in order to compute, but they must be brought to the left step by step, cell by cell.) Which is the influence of the size of antiport rules on the computing power? What about considering deterministic cell Turing machines? What does correspond to a linearly bounded automaton? (A possibility is to consider two types of non-differentiated cells, first with a multiset w inside, then, to the right of them to the infinity, empty.) What a push–down automaton could mean in this framework?

T The previous question can be extended to cellular automata, which also have "cells" which . . . are not cells. How a cellular automaton with cell–membranes arranged on a grid, containing multisets and exchanging objects (horizontally or vertically) by using antiport rules, can be defined remains to be found. Because we do not create or destroy objects, we either have to confine ourselves to playing a purely combinatorial game, or we need a supply of objects (for instance, by considering the neighborhood of a finite region as non-differentiated, like in a cell Turing machine, but non-empty).

U The mentioning of cellular automata suggests a question famous in this area: reproduction. In general, the question is how can we produce "automatically" a P system identical to a given system, or similar to it, whatever that means. The motivation is not only related to the reproduction question in cellular automata, but also to complexity issues: when solving a hard problem, we have to construct a system or a family of systems, in polynomial time, by means of a Turing machine, and the question was formulated whether this task can be accomplished by a P system, so that the whole process would become uniform (with all phases based on using P systems).

As formulated here, the problem is very general, so let us try to be more specific. Consider a set R of rules handling multisets and membranes. Starting from a given configuration and using the rules from R we get a family of configurations. Adding to them rules for objects evolution and for handling the membranes, we get a family of P systems. For certain sets of rules R, we can start from very simple configurations and, as above, we can produce the whole family of P systems of a certain type – details can be found in [7].

Now, this suggests a specific problem related to the reproduction general issue: start from a "seed" system/configuration, C_0, and apply the rules from a given set R, until obtaining a system/configuration C; is it possible to continue the computation (the use of rules from R) in such a way to obtain one more system/configuration C? The idea is to produce (maybe non-deterministically) the first copy of C, but to continue by producing copies of the same configuration/system C. To this aim either the seed C_0 and the rules should be arranged in such a way that the process is deterministic, or we have to remember somehow the way C was produced (its "genome") and to use this information from now on in order to obtain further copies of C.

Of course, a counter can be introduced in C_0, such that only a limited number of copies of C are produced. A relaxed version of this problem is to look not for identical copies of C but for similar copies, a case where a similarity relation should be first considered.

V The idea of starting from a set of configurations and to use a set of rules in order to produce a family of configurations directly reminds the style of the AFL (Abstract Families of Languages) theory, which can be imitated also in our context. Some steps in this direction were done in [7], but this research area is so large that it deserves to be mentioned here even without giving details, just to call the reader's attention about it.

X There is a warm debate in modelling biological systems, whether or not the used models should use continuous or discrete mathematics. Continuous mathematics means in general (systems of) differential equations. There are many pros and cons for both types of mathematics, and, as usual, probably the truth is placed *in media res*. Still, the multiset rewriting in terms of P systems seems to be a very useful tool, at least because it is easily understandable, easily scalable, easily programmable – three features which cannot be claimed also for systems of differential equations. On the other hand, there are situations where the reality seems to be better described by continuous parameters (this is the case for concentrations/gradients, probabilities, etc). Bringing some continuous mathematics inside membrane computing was attempted several times – a recent example is [24] (where a solid application related to cancer investigations can be found). Still, this topic of developing a 'theory" of non-discrete P systems requests (and deserves) further investigations.

There are both theoretical and practical issues here. Working with real numbers raises problems concerning the computability of the system components. Then, how much continuum should be introduced? Multisets with real multiplicities – in regions and/or in the rules, using each rule a real number of times (like in [24])? What else? Can we work with a continuous system as it is, or we have to approximate it by a discrete one? With what degree of approximation? A trade–off should exist here between the degree of approximation and the difficulty of handling the system (in particular, of programming and simulating it on a computer). These problems seem to be closely related to handling fuzzy P systems – see approaches (and further references) to this topic in [1]. In particular, one can find in [1] a list of specific research topics, [8].

Y Related to the previous question: can a direct passage be found between systems of differential equations and multiset–rewriting P systems? Of course, a price should be paid, maybe in terms of losing some information, or some accuracy of handling the information. However, the possible gain seems to be worth considering: understandability, scalability, programmability. If the problem does not have a solution in general, are there particular classes of systems of differential equations which can be faithfully reformulated as sets of multiset–rewriting rules? (Are there real life problems which can be described by such equations?)

This kind of approach can have several motivations. For instance, because there are many models based on differential equations, if an associated P system can describe in a similar way the considered piece of reality, then we can continue by using the P system; this means, for instance, that we can use the same type of systems for describing other real phenomena, or we can extend the system, adding new parameters and new rules (which is not an easy task in terms of differential equations).

W One more general issue, related to the way of defining the output of computations in a (symbol–objects) P system. Up to now there were used several ideas: counting the objects from a specified region or leaving (entering, in the case of P automata) the system, considering the sequence of objects leaving (entering) the system, taking the trace of a specified object, or the tree itself describing the configuration in the end of a computation. What else? Here are a few proposals, without knowing which of them make sense and which not.

The main idea is not to use a *support* for information, such as the multiplicity of objects, but to take *events* – much like in the case of trace languages. (The traces have the drawback that we need at least as many membranes as the cardinality of the alphabet of the language we want to describe.) For instance, let us distinguish some special objects and count the number of times when they meet each other in any membrane of the system. Precise pairs of objects can be given in advance. We can also consider the sequence of events of this type (as usual, accepting all permutations when several events take place simultaneously). Actually, we have two possibilities: to count only the moment when two distinguished objects come together in a region (hence not also the subsequent steps when they remain both in the same region), or to count all moments when the two objects are together, even if they do not move in the meantime.

Then, we can consider as events the use of certain rules – which leads to a sort of control word, or Szilard word, associated with a computation.

Another idea is not to count the events themselves, but to look for some parameters (related to events). One possibility is to count the number of steps elapsed in between two specified events. For instance, the very length of a computation (the number of steps from an initial configuration to a halting one) can be taken as the number computed by the system. Or, we can consider a copy of a special object d, and take as the result of the computation the number of steps elapsed from the beginning of the computation until sending the object d to the environment. Similarly, we can take two copies of d and count the number of steps between sending out the two objects. Of course, any type of event can be considered, but the definition should be carefully adapted (a difficulty appears, for instance, when the event is repetitive; which interval should be considered? the first, the last, all, none?).

Z Let us end with another general topic, related to control theory. Consider a P system describing a real system; we start from an initial configuration C_0 and evolve non-deterministically. Assume that we can distinguish (how? this can be done in an easy way from a computational point of view, for instance, by using a finite state multiset automaton) two classes of configurations, "bad" and "good" (their sets are supposed disjoint), and that C_0 is a good configuration. Asking whether or not the system will ever reach a bad configuration is a reachability problem. The configurations (reachable from C_0) can be of several types, depending on the configurations which can be immediately reached from them: (i) leading to only good configurations, (ii) leading to only bad configurations, (iii) leading to both good and bad configurations. This can happen both for good and bad configurations. In case (iii) we have a branching which looks dramatic for the "fate" of the system. Similarly bad are the moments when the system passes from good to bad configurations. Can the sequence of such important moments be described (by a string)? Which is the language of these crucial steps? If any trajectory which reaches a bad configuration remains forever in the bad region (think to the case of a serious illness), then it is important to find the moments when the system passes from a good configuration to a bad one. Which is the first/last moment when we can predict the evolution of the system?

Then, we can ask for "healing" possibilities. If from a configuration C there is a continuation to a bad configuration, is it possible to inject certain objects in the system so that all continuations will lead to good configurations? Which is the first/last moment when this can be done, which is the smallest multiset to insert? Can these questions be answered algorithmically? (Not for universal systems, so that we have to look for restricted classes.)

On the same lines, let us call a system *curable* if there is a finite multiset M such that, adding to the system in each time unit a sub-multiset of M (maybe the empty one), then the trajectory passes always through good configurations. Given a system Π, can we decide whether it is curable? If a system is curable, can a (minimal) multiset M as above be effectively found? Which can be a sequence of sub-multisets of M which cures the system?

The medical interpretation is obvious here. When a check should be made, when and what "injection" should be provided? A general cost of these actions could be considered. Further related questions are possible – but the first step should be a more precise formulation of the problem. We feel that there is a lot of work to be done in this respect, both from a theoretical and a practical point of view.

No more letters, no more problems... Not in this list, but there are numerous open problems, in many cases explicitly formulated, in the membrane computing literature, hence the reader can easily complete this list up to one with one hundred twenty–six entries. Or, on the contrary, can shorten the list, by providing answers to some problems. Either of these reactions is much welcome.

Acknowledgements

Several of the previous problems, or related issues, have appeared in papers by or were discussed with A. Alhazov, D. Besozzi, G. Ciobanu, E. Csuhaj-Varju, A. Di Nola, R. Freund, M. Gheorghe, M.A. Guttiérez–Naranjo, O.H. Ibarra, N. Krasnogor, S.N. Krishna, M. Margenstern, T.Y. Nishida, L. Pan, A. Păun, M.J. Peréz-Jiménez, G. Rozenberg, D. Sburlan, P. Sosik, G. Vaszil, S. Verlan, and others, to whom I am very much indebted, and who might have already answers to some of these problems (this is a suggestion to the reader to contact them for details, leaving to me the role of a catalyst. . .).

References

1. R. Alberich, J. Casasnovas, M. Llabrés, J. Miró–Juliá, J. Rocha, F. Rosselló, eds.: *Brainstorming Workshop on Uncertainty in Membrane Computing.* Universitat de les Illes Baleares, Palma de Mallorca, November 2004.
2. A. Alhazov: Personal communication, 2005.
3. A. Alhazov, R. Freund, Gh. Păun: P systems with active membranes and two polarizations. In [22], 20–36.
4. A. Alhazov, C. Martín-Vide, L. Pan: Solving a PSPACE-complete problem by P systems with restricted active membranes. *Fundamenta Informaticae*, 58, 2 (2003), 67–77.
5. M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.: *Proceedings of the Brainstorming Week on Membrane Computing; Tarragona, February 2003.* Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003.
6. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing.* Springer-Verlag, Berlin, 2005.
7. E. Csuhaj-Varju, A. Di Nola, Gh. Păun, M.J. Pérez-Jiménez, G. Vaszil: Editing configurations of P systems. In the present volume, 131–154.
8. A. Di Nola, Gh. Păun, M.J. Pérez-Jiménez, F. Rossello: (Imprecise topics about) Handling imprecision in P systems. In [1], 1–10.
9. R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Sci.*, 330, 2 (2005), 251–266.
10. R. Freund, M. Oswald: Tissue P systems with symport/antiport rules of one symbol are computationally universal. In *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez–Naranjo, Gh. Păun, M.J. Pérez–Jiménez, eds.), Fénix Editora, Sevilla, 2005, 187–200.
11. O.H. Ibarra: On determinism versus nondeterminism in P systems. *Theoretical Computer Science*, to appear (available at `http://psystems.disco.unimib.it`).
12. N. Jonoska, M. Margenstern: Tree operations in P systems and λ-calculus. *Fundamenta Informaticae*, 59, 1 (2004), 67–90.
13. S.N. Krishna, Gh. Păun: P systems with mobile membranes. *Natural Computing*, to appear.
14. T.Y. Nishida: An application of P systems: 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, 109–112.

15. T.Y. Nishida: Membrane algorithms: Approximate algorithms for NP-complete optimization problems. In [6], 301–312.
16. Gh. Păun: Computing with membranes (P systems): Twenty six research topics. CDMTCS Research Report 119, Febr. 2000, Auckland Univ., New Zealand (www.cs.auckland.ac.nz/CDMTCS).
17. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
18. Gh. Păun: Further open problems in membrane computing. *Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University, 2004 (BWMC2004), 354–365.
19. Gh. Păun: Membrane computing: Some non-standard ideas. In *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950, Springer-Verlag, Berlin, 2004, 322–337.
20. Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodriguez-Paton: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64, 1-4 (2005), 353–367.
21. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núnẽz: Tissue P systems with cell division. In [22], 380–386.
22. Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.: *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, February 2004.* Technical Report 01/04 of Research Group on Natural Computing, Sevilla University, Spain, 2004.
23. M.J. Pérez-Jiménez: Personal communication, 2005.
24. M.J. Pérez–Jiménez, F.J. Romero–Campero: Modelling EGFR signalling network using continuous membrane systems. *Third Workshop on Computational Methods in Systems Biology*, Edinburgh, 2005.
25. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini: Computationally hard problems addressed through P systems. In [6].
26. P. Sosik: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2, 3 (2003), 287–298.