Towards "Fypercomputations" (in Membrane Computing)

Gheorghe Păun

Institute of Mathematics of the Romanian Academy,
P.O. Box 1-764, RO-014700 Bucharest, 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
george.paun@imar.ro, gpaun@us.es

Abstract. Looking for ideas which would lead to computing devices able to compute "beyond the Turing barrier" is already a well established research area of computing theory; such devices are said to be able of doing *hypercomputations*. It is also a dream and a concern of computability to speed-up computing devices; we propose here a name for the case when this leads to polynomial solutions to problems known to be (at least) **NP**-complete: *fypercomputing* – with the initial **F** coming from "fast".

In short: *fypercomputing means going polynomially beyond* **NP**.

The aim of these notes is to briefly discuss the existing ideas in membrane computing which lead to fypercomputations and to imagine new ones, some of them at the level of speculations, subject for further investigation.

Keywords: Turing computing, Hypercomputing, Membrane computing, Complexity.

1 Foreword: Jürgen the Fast

These pages are dedicated to the 65th birthday of Jürgen Dassow. The age of retirement! I have met him in 1983 (it was my first trip out of Romania), we collaborated a lot, we met periodically in the framework of IMYCS - International Meeting of Young Computer Scientists, at Smolenice Castle, Slovakia. A young computer scientist is now retiring! He is maybe not the first, but I am not aware of others (to whom I was close enough). I feel this as an end of an epoch... The romantic time of grammars and automata, of sequential computing, of theory for the sake of theory, with aesthetic motivation. I remember the time spent in Magdeburg as a Humboldt fellow in 1991-92 and after that as a visiting researcher. Jürgen was, after 1993, the rector of the university. A very dynamical period, after the unification of Germany, with a lot of bureaucratic work – but with Jürgen switching in an amazingly quick way from administrative matters

H. Bordihn, M. Kutrib, and B. Truthe (Eds.): Dassow Festschrift 2012, LNCS 7300, pp. 207–220, 2012.

to scientific matters. I have never seen such a distributed and fast mind – and this fits with the topic of my notes. I cannot imagine Jürgen "doing nothing", so I am wishing to him, young pensioner, a long and active life!

2 Introduction – From Hyper... to Fyper...

The aim of this note is to briefly discuss an important research topic in computer science in general, in membrane computing in particular, namely ways to speed-up computations. The goal is to obtain devices able to significantly improve from this point of view, e.g., to solve intractable problems (usually, **NP**-complete or even harder) in a feasible time (usually, polynomial). Our constant framework is membrane computing (the reader is supposed to be familiar with the domain, or (s)he can consult [17], [19], [21], and the domain website from [30]), but some considerations/speculations are more general. (By the way, Jürgen is a co-author of one of the earliest papers in membrane computing, [7].)

The model we have in mind is that of hypercomputations, already with a large literature (see, e.g., [6], [9], [25], or the recent survey from [27]). The goal is to imagine computing machineries able "to compute the uncomputable", to compute more than Turing machines. More than a dozen of ideas were proposed and proved to reach the above goal: oracles (already considered by Turing), introducing real numbers in the device, accelerating the functioning of machines, using ingredients of an analogical nature and so on.

Also in membrane computing there were reported attempts of this kind. We mention here the papers [4] and [26]. The first one passes beyond Turing by means of acceleration (we come back below to this idea), the latter by constructing "lineages" of P systems, following the model of [28].

It should be noted that there also are people who do not believe in hypercomputation, refuting it as a circular trick (basically, one introduces uncomputable components in a computing device which is then shown to compute more than a Turing machine...); one of the most explicit voices in this respect is Martin Davis – see, e.g., [8].

We are looking here not for the *power* of computing devices, but for their *efficiency*, and we propose the term *fypercomputation* to name the case when a device can solve an intractable problem in a polynomial time. Surprisingly enough, this research vista has not received a similar attention as hypercomputation; the literature of the latter one is much more extended (more explicit), there were organized debates about it, workshops and special sessions in general conferences. Complexity theory discusses many speed-up procedures, but not at the level of "breaking the **NP** barrier", to put it in the style of "breaking the Turing barrier". The time-space trade-off is a common sense in algorithmics, but using an exponential space is not considered a serious candidate for fypercomputations – excepting the recent natural computing area, molecular (DNA) computing in special.

It is true, there are many papers which can be put under the flag of fypercomputing. They exploit ideas from physics, such as [24] (the abstract of the paper

is worth recalling: *We propose to "boost" the speed of communication and computation by immersing the computing environment into a medium whose index of refraction is smaller than one, thereby trespassing the speed-of-light barrier*), propose analogical computations, such as [2] and [3]. The area of DNA computing is full of such ideas, its very beginning is of this kind, [1]. The starting point is the fact that the DNA molecules are a very compact support of information, one can operate with bits stored at a molecular level, hence in a small space (in a test tube) we can have a huge number of molecules. "Huge" however, has no precise mathematical meaning; when we need "exponentially many" molecules, what is perfect in theory becomes unfeasible in a lab...

But not this is the position we adopt here: however "unrealistic" (for current knowledge, be it physical, biological or of another nature), we look for "nice" ideas able to lead to fypercomputations. And membrane computing abounds already in such investigations.

3 Ways towards Fyper... in Membrane Computing

The main practical goal of natural computing is to learn computing ideas from nature and to implement/use them in computer science, starting with a better use of the existing computers and ending – the dream of DNA computing and of quantum computing, at least – with the construction of computers of a new kind, maybe using new materials. In all cases, the aim is to be more efficient, especially with respect to the duration of computations.

Membrane computing started with a more theoretical goal in mind: just learn interesting ideas for computability from the structure and the functioning of the living cells.

The basic computing models (called *cell-like P systems*) considered in this area consist of an hierarchical arrangement of *membranes*, delimiting *compartments* (also called *regions*) where multisets of *objects* (symbols from a given alphabet) are placed; these objects evolve according to *evolution rules* also associated to regions.

The membranes are labeled with symbols in a given alphabet (in most cases we use natural numbers). Each region is precisely identified by its upper membrane, hence we can identify regions by means of the labels of the membranes which delimit them. The external membrane, the one separating the cell from the *environment* is called the *skin* membrane; a membrane without any membrane inside is said to be *elementary*. We also call region the environment of the system (and usually we label it with 0).

The rules are of a multiset rewriting type (corresponding to biochemical reactions taking place in a cell) or of other forms, in general, inspired from biology (transport across membranes in the form of symport or aniport, membrane division, membrane creation, etc.). Starting from an initial configuration of the system (with initial multisets of objects associated with the regions) and using the rules in various ways (the most investigated one is the nondeterministic maximally parallel way), one passes from a configuration to another one. Such

a sequence of *transitions* among configurations is called a *computation*. With a *halting* computation (one reaching a configuration where no rule can be applied) we can associate a result, for instance, as the number of objects present in a given region. We will give some further details in the next sections, this quick description of a P system and of its functioning should be sufficient for understanding the general discussion which follows.

The class of cell-like P systems is the basic one (and most investigated) in membrane computing. Several other types of similar devices were considered. For instance, instead of symbol objects, one can consider string objects (then, the rules should be string processing rules, such as rewriting, the splicing from DNA computing, etc.), or, instead of an hierarchical arrangement of membranes (hence described by a tree), we can place the membranes in the nodes of an arbitrary graph – one obtains in this way the *tissue-like P systems*. Also P systems inspired from the neurons architecture and functioning were introduced – a class of a particular interest is that of *spiking neural P systems* (SN P systems, for short), where *neurons* (membranes) placed in the nodes of a graph communicate to each other by means of *spikes*, electrical impulses of identical shapes (hence, the system uses only one object, the spike, present in various numbers of copies in different neurons) which are processed by specific *spiking* rules. We refer to [13] and to the corresponding chapter of [21] for details.

Besides the computing power of P systems (in comparison with classic computing devices, such as Turing machines and their restrictions), also the efficiency issue was addressed, and, in order to speed-up computations, the first proposal was to make use of the biological operation of cell division. In this way, P systems with active membranes have appeared, [18], and they proved indeed to speed-up computations enough for solving **NP**-complete problems in a polynomial time. (The typical rules are of the form $[a]_h \to [b]_h[c]_h$, where a, b, c are objects and h is a label; the membrane h, containing the object a and maybe other objects, is divided into two new membranes, with the same label h; in the first copy, a is replaced by b, in the second one it is replaced by c; any objects different from a are replicated in the two new membranes. Clearly, the new membranes with label h use the same set of rules as the former membrane h; otherwise stated, the label of a membrane identifies the set of rules which can be applied to it.)

An important point should be made here. P systems (with rules of the form $a \to u$, where a is an object and u is a multiset of objects) are able to create exponentially many objects in a linear time: using the rule $a \to aa$ in the maximally parallel manner for n times, we get 2^n copies of a. However, such an exponential workspace is not enough in order to essentially speed-up computations: the so-called Milano Theorem from [29] says that a P system (with rules as above, but without membrane division) can be simulated by a Turing machine with a polynomial overhead. Membrane division produces an exponential workspace, but also introduces some *organization* of it, some *localization*. In a specified membrane, specific rules are used, which is not possible without separating the objects among "protected reactors" – like in biology (compartments with specific chemicals evolving according to specific sets of reactions).

This lesson, of localization, with specific "chemistry" taking place in each compartment, is applied also in the case of P systems with membrane creation, [16], where rules of the form $a \to [b]_h$ are used, transforming an object a into a new membrane, with label h and containing the object b (as above, the label of the membrane identifies the set of rules to apply to its objects). This time, the aspect mentioned above, i.e., the power of the localization, is still more visible: The rules for creating membranes produce only a linear number of membranes, not an exponential one, as in the case of division, but the exponential space is produced by object evolution rules (such as $a \to aa$).

There are many other important details in this area, for instance, additional ingredients which have (or not) an influence on the efficiency of P systems with membrane division or membrane creation (for instance, membrane polarization, membrane dissolution, division of only elementary membranes or also of non-elementary membranes). In most cases, polynomial solutions to **NP**-complete problems – often, also of **PSPACE**-complete problems – are obtained. Fypercomputing! (The list of computationally hard problems addressed in terms of membrane computing is very large, ranging from decidability problems, such as `SAT` and `QSAT`, to numerical problems – e.g., counting various parameters in a graph; a chapter of [21] is devoted to this research direction, and the reader is refereed to it for details.)

Further two similar ideas were investigated in membrane computing, both of them related to the previous two. The first one is based on string replication, [5], for P systems with string objects (the rules are of the form $a \to u_1||u_2$, where a is a symbol, u_1, u_2 are strings; when rewriting a string xay by such a rule we obtain two strings, xu_1y and xu_2y, maybe placed in two different membranes, because the strings u_1, u_2 can also have associated target indications). Replicating a string looks like dividing a membrane: the symbols of each string remain together, hence "localized", like being encapsulated in separated membranes. Fypercomputations are again obtained (in [5] one shows how `SAT` can be solved in polynomial time in this framework).

A different approach is that of considering arbitrarily large *pre-computed resources*. For instance, we can assume as given in advance, for free (from the point of view of the computing time), a spiking neural P system (this is the case of [14], [15]), with arbitrarily many neurons and synapses, but with a small (finite) number of spikes inside and having a regular structure; otherwise stated, the given system is large, but it cannot contain more than a bounded amount of information, not enough, for instance, to encode the solution to the problem we want to solve. Then, we plug in a code of the (instance of the) problem to solve, in the form of a bounded number of spikes (a bounded number of bits), and this activates the arbitrarily large "hardware", which eventually provides the solution in a polynomial time.

This last idea is only briefly investigated in membrane computing. Issues related to the conditions to be imposed to the given pre-computed resources should be further considered.

Anyway, we count already four ideas, all of them having biological motivations (maybe better: all of them suggested by biological facts), which lead to fypercomputations in the membrane computing area.

There is a large bibliography of this direction of research. We refer here only to the survey paper [22], where the basic ideas of computational complexity for P systems can be found, and to the chapter of [21] devoted to this topic, [23].

4 Further Ways towards Fypercomputations in Membrane Computing

In membrane computing there are several versions of the previously mentioned speed-up tools, for instance, operations of separating membranes according to their contents, budding membranes, etc., but they have the same philosophy and the same functions as membrane division, so we do not count them separately here, but we look for ideas essentially different.

(1) The first candidate is the *acceleration*. The idea of an accelerated Turing machine is old in computer science: imagine that the machine is "clever", it learns from its own functioning, in the following way; after performing a step in a time unit, it performs better for the second step, which is completed in half of the time necessary for the first step – and so on, at each step halving the time with respect to the previous step. If the first step takes one time unit, then the second one takes 1/2 time units, the third one 1/4 and so on, hence in two time units the computation ends.

Important: we have here two clocks, an internal one, of the machine, and an external one, of the observer. The internal clock is faster and faster, so that the computation ends in two time units *measured by the external clock*, that of the observer/user.

Accelerated Turing machines can solve the halting problem, hence they compute what usual Turing machines cannot. See references in [4], where the idea is extended to P systems: It is known from biology that nature creates membranes also for enhancing the reactions inside (if the reactants are closer, then the probability to collide and react is higher). If "smaller is faster", then take it as in accelerated Turing machines: the reactions taking place in a membrane placed inside another membrane are twice faster than in the parent membrane. Using membrane creation rules, create then membranes inside membranes, in an unbounded hierarchy, which means faster and faster towards the "center" of the membrane structure. Like for accelerated Turing machines, such P systems were proved to "compute the uncomputable".

Well, then why not using this trick also in complexity sense? Natural enough, but Martin Davis is smiling ironically: we accelerate in order to speed-up... In two (external) time units we solve any problem, whatever complex it is (remember that most classes of P systems are Turing complete, they can do whatever a Turing machine can do).

A way to make the things interesting is to accelerate only parts of a P system, thus having several levels of time speed. For instance, imagine a P system

with several internal membranes, all of them governed by the same clock, but some, say, elementary membranes being accelerated: for each of them we have a constant k_h (h is the label of the membrane) and, after introducing a "subproblem" in membrane h (e.g., a string x and a language L – by means of a grammar), we get the answer (the fact whether or not $x \in L$) in at most k_h time units (as measured by the global clock). Not all types of P systems are universal when using only one membrane, so, for such cases, the local acceleration can be indeed of interest (i.e., non-trivial). What other restrictions to consider remains as a research topic.

(2) For instance, we can accelerate not elementary membranes, but rules: a given rule takes one time unit for the first application, half for the second application, and so on. If all rules are accelerated, we go back to globally accelerated systems, but if we allow only to a bounded number of rules to get faster in this way, it is possible to obtain interesting results.

(3) Maybe also objects can be accelerated: the descendants of an object react faster than the father object, irrespective which are the rules which act on them and irrespective of the membranes where they are.

(4) The previous ideas suggest a speculation of a *science-fiction* type, but let us place it here, the context is speculative enough. Namely, we mentioned before (at least) two clocks, an external one, of the observer (or of the higher membranes in the structure) and the local clock(s), of the accelerated element, membrane, rule, object. Always, the inner clock is (much) faster than the external one, it performs sometimes an exponential number of steps while the external one only ticks once. We can then imagine that the inner time is orthogonal to the external time, hence the time has a 2D structure! Is this too much with respect to the "classic" time, in general interpreted as linear (or circular, spiral, in various philosophies)? As a speculation, I would not refuse to think about this.

So: what about considering the time as a plane, not as a line, with the observer only sensing one dimension of it, but with the possibility of some "processors" to run along the other dimension, doing this at-no-time for the observer? (After all, the current models of the universe, the so-called membrane theory – having nothing common with membrane computing! –, which has replaced/generalized the string theories, deals with eleven dimensions, enough for "lateral" times...)

Well, passing from this speculation to precise models, to symbols, algorithms and theorems, is another story... (but what else are the oracles than processors working in a "lateral" time – and space?).

5 P Systems with ω-Populations of Objects

(5) Let us come back to the ground, namely to the recently introduced *reaction systems* (we call them *R systems*) – see [10], [11], [12]. An attempt to bridge the two research areas, P systems and R systems, was done in [20]; the present section can be seen as a continuation of this attempt.

Both areas deal with populations of reactants (molecules) which evolve by means of reactions, with several basic differences. Most of these differences are not mentioned here (e.g., the compartmental structure of models in membrane computing versus the lack of membranes in reaction systems; the focus on evolution, not on computation, in reaction systems; the unique form of rules in reaction systems and so on), and we recall the two basic ones, the crucial postulates of R systems, in the formulation from [10]:

The way that we define the result of a set of reactions on a set of elements formalizes the following two assumptions that we made about the chemistry of a cell:

(i) We assume that we have the "threshold" supply of elements (molecules) – either an element is present and then we have "enough" of it, or an element is not present. Therefore we deal with a qualitative rather than quantitative (e.g., multisets) calculus.

(ii) We do not have the "permanence" feature in our model: if nothing happens to an element, then it remains/survives (status quo approach). On the contrary, in our model, an element remains/survives only if there is a reaction sustaining it.

Bringing the first assumption in membrane computing is a way to obtain fypercomputations! This, however, is not very surprising: if each object is present in "enough" copies, this means already that we have an arbitrarily large workspace. The only problem is how to use this unstructured space for efficient computations.

Because we want to get a little bit more technical here, we introduce some formal prerequisites.

Given an alphabet O of symbols called here *objects*, we denote by O^* the set of all strings over O, the empty string λ included, and by $O^+ = O^* - \{\lambda\}$ the set of all non-empty strings over O. A multiset-rewriting rule (over O; we also say *evolution rule*) is a pair (u, v), written in the form $u \to v$, where u and v are multisets over O (given as strings over O). The rules are classified according to the complexity (of their left hand side). A rule with at least two objects in its left hand side is said to be *cooperative*. The result in Theorem 1 is given for such rules; a restrictive type of rules is that of *non-cooperative ones*, of the form $u \to v$, with u being an object in O.

Now, a *cell-like P system* (of degree m) is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out}),$$

where O is the alphabet of objects, μ is the membrane structure (with m membranes), given as an expression of labeled parentheses, $w_1, \ldots, w_m$ are (strings over O representing) multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of evolution rules associated with the regions of μ, and i_{in}, i_{out} are the labels of input and output regions, respectively, $i_{in} \in \{1, 2, \ldots, m\}, i_{out} \in \{0, 1, 2, \ldots, m\}$; if $i_{out} = 0$, this indicates that the output is obtained in the environment. If the system is used in the

generative mode, then i_{in} is omitted, and if the system is used in the accepting mode, then i_{out} is omitted. The number m of membranes in μ is called the *degree* of Π.

The rules in sets R_i are of the form $u \to v$, as specified above, with $u \in O^+$, but with the objects in v also having associated *target indications*, i.e., $v \in (O \times \{here, out, in\})^*$. After using a rule $u \to v$, the objects in u are consumed, and those in v are produced; if $(a, here)$ appears in v, then a remains in the same compartment of the system where the rule was used, if (a, out) is in v, then the object a is moved immediately in the region surrounding the compartment where the rule was used (this is the environment if the rule is used in the skin region), and if (a, in) is in v, then a is sent to one of the inner membranes, nondeterministically chosen (if there is no membrane inside the membrane where the rule is meant to be applied, then the use of the rule is forbidden). The indication *here* is omitted, we write a instead of $(a, here)$.

The rules are used in the nondeterministic maximally parallel manner: in each membrane, a multiset of rules is applied such that there is no larger multiset of rules which is applicable in that membrane.

In the generative mode, the result of a computation consists of the number of objects in region i_{out} in the moment when the computation halts, i.e., no rule can be applied in any membrane of the system. In the accepting mode, a number is introduced in the region i_{in}, e.g., in the form of the multiplicity of a given object, and, if the computation halts, then this number is accepted. A P system can also be used in the computing mode, with a number introduced in region i_{in} and the result obtained in region i_{out}, in the moment when the computation halts. P systems can be also used for solving decidability problems: an encoding of a problem (of an instance) is introduced in membrane i_{in}, in the form of a multiset of objects, and the answer, YES or NO, is obtained, for instance, by halting, or by sending out of the system a distinguished object *yes* or *no*.

Note that in the previous definitions multisets play a crucial role, objects not evolving by a rule remain unchanged, and that always successful computations are defined by halting – all these are essential differences with respect to R systems.

Consider now a P system with some of its objects being present in arbitrarily many copies, like in R systems. More exactly, we consider P systems which contain certain distinguished elementary membranes, whose objects are present in arbitrarily many copies (for instance, if an object a is introduced from outside in such a membrane, then inside the membrane it becomes a^ω; it enters as a single copy, and immediately multiplies inside to arbitrarily many, like in reaction systems).

Let us call such a system an ωP system.

The arbitrary multiplicity of objects introduces an important change in the functioning of a usual system. For instance, if we have the objects a, b, c in a distinguished membrane, together with the rules $ab \to d, ac \to e$, then both these rules can be (and should be) applied, because we have *enough* copies of a for both rules; we obtain d, e, with *all* copies of a, b, c being consumed. If also an object f is present together with a, b, c, then it remains unchanged.

We do not adopt here also the second assumption from the definition of R systems, because then all objects from a halting configuration vanish. Please note that the "easy solution" of introducing dummy rules of the form $f \to f$ for all objects which cannot evolve otherwise does not work either: the computation will never stop.

This apparently innocent change in the structure of P systems, i.e., considering objects with ω multiplicity, is able to speed-up a P system to the level of fypercomputations.

Theorem 1. SAT *can be solved (in a uniform way) in a polynomial time by an* ωP *system.*

Proof. Let us consider the SAT problem for n variables, $x_1, x_2, \ldots, x_n$, and m clauses. We consider x_l and $\neg x_l$, $1 \leq l \leq n$, as symbols, we denote by Lit their alphabet, and we also denote $Val = \{t_i, f_i \mid 1 \leq i \leq n\}$. If $\alpha \in Lit$, then $\neg\alpha$ is its negation (e.g., $\neg(\neg x_i) = x_i$).

An instance $\gamma = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ of $\mathtt{SAT}(n, m)$, with $C_i = y_{i,1} \vee y_{i,2} \vee \ldots \vee y_{i,k_i}$, for $y_{i,j} \in Lit, 1 \leq j \leq k_i$, is encoded as

$$code(\gamma) = y_{1,1}^{(1)} \ldots y_{1,k_1}^{(1)} y_{2,1}^{(2)} \ldots y_{2,k_2}^{(2)} \ldots y_{m,1}^{(m)} \ldots y_{m,k_m}^{(m)}.$$

We now construct the following ωP system (the unique elementary membrane is the distinguished one, the region where the objects are present with ω multiplicity):

$$\begin{aligned}
\Pi &= (O, \mu, w_1, w_2, R_1, R_2, 1, 0),\\
O &= \{\alpha^{(j)} \mid \alpha \in Lit, 1 \leq j \leq m\} \cup Val\\
&\quad \cup \{a, \mathtt{yes}, d, d_1, d_2, \ldots, d_{m+1}\} \cup \{\langle aw\rangle \mid w \in Val^*, |w| \leq m\},\\
\mu &= [\,[\quad]_2\,]_1,\\
w_1 &= a d_{m+1},\ w_2 = \lambda,\\
R_1 &= \{a \to (\langle a\rangle, in),\ d_1 \to (d, in),\ \mathtt{yes} \to (\mathtt{yes}, out)\}\\
&\quad \cup \{x_i^{(1)} \to (t_i, in),\ \neg x_i^{(1)} \to (f_i, in) \mid 1 \leq i \leq n\}\\
&\quad \cup \{x_i^{(j+1)} \to x_i^{(j)},\ \neg x_i^{(j+1)} \to \neg x_i^{(j)} \mid 1 \leq i \leq n, 2 \leq j \leq m\}\\
&\quad \cup \{d_{j+1} \to d_j \mid 1 \leq j \leq m\},\\
R_2 &= \{\langle aw\rangle x_i \to \lambda \mid \neg x_i \in w, w \in Val^+, 1 \leq i \leq n\}\\
&\quad \cup \{\langle aw\rangle \neg x_i \to \lambda \mid x_i \in w, w \in Val^+, 1 \leq i \leq n\}\\
&\quad \cup \{\langle aw\rangle\alpha \to \langle aw\rangle \mid \alpha \in w, \alpha \in Val, w \in Val^+\}\\
&\quad \cup \{\langle aw\rangle\alpha \to \langle aw\alpha\rangle \mid \alpha \notin w, \neg\alpha \notin w, \alpha \in Val, w \in Val^*\}\\
&\quad \cup \{\langle awd\rangle \to (\mathtt{yes}, out) \mid w \in Val^+\}.
\end{aligned}$$

For an easier understanding, we also present this system in a graphical form, in Figure 1 (besides the membranes and the objects present in the initial configuration, the rules from each regions are specified).

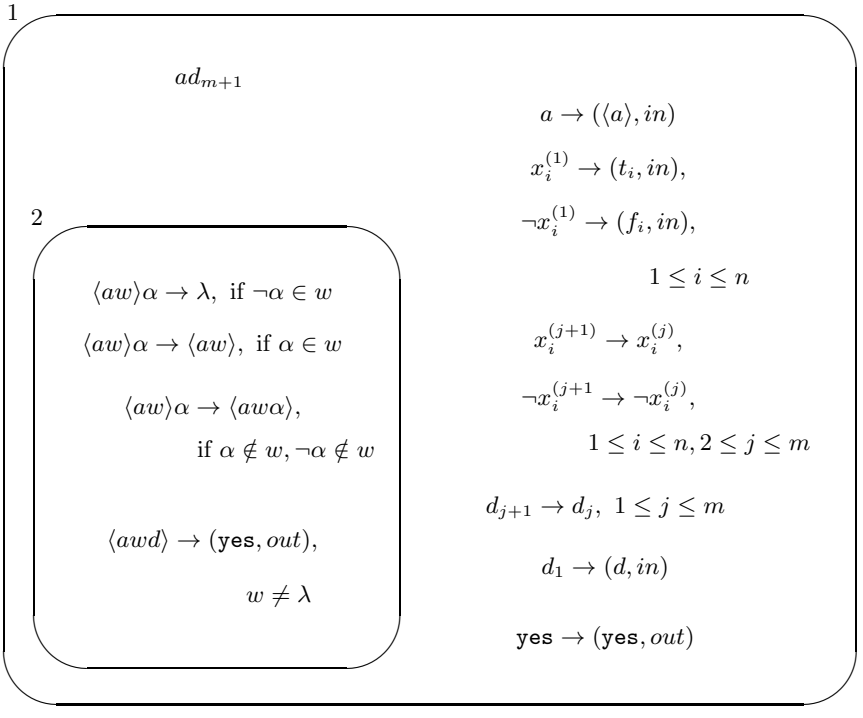

Fig. 1. The ωP system from the proof of Theorem 1

The computation of Π starts after introducing the multiset $code(\gamma)$ in the skin membrane, for a given instance γ of the $\mathtt{SAT}(n, m)$ problem. The "seed" object a enters immediately membrane 2, in the form $\langle a \rangle$, together with the truth values which make true the first clause (the literals in clause C_1 are present in $code(\gamma)$ with superscripts 1). All other objects evolve in the skin membrane, those associated with literals in clauses C_2 to C_m of γ decreasing by one the superscript, d by decreasing by one the subscript.

In the inner membrane we have (at an arbitrary step) several multisets aw in the bracketed form $\langle aw \rangle$ (hence interpreted as symbol objects); initially, $w = \lambda$. Each multiset w contains truths values of variables which make true the clauses of γ, starting with the first clauses in the ordering $C_1, C_2, \ldots$. When the truth values which render true a further clause enter membrane 2, they are checked against the existing truth values, those which made true the previous clauses. It is important to note that *each of the existing sets of truth values "react" with each newly introduced truth value*, in the style of reaction systems, because we have arbitrarily many copies of each object present in membrane 2. If a previous set of truth values w falsifies the new clause, then the object $\langle aw \rangle$ disappears. If it contains a truth value which renders true the new clause, then it survives. When neither of these situations holds, then we add to w a truth value which makes true the new clause (without falsifying the previous clauses).

The process continues for each of the m clauses. In the end, the "checker" d enters membrane 2. If it finds here at least one object $\langle aw \rangle$ with $w \neq \lambda$, this means that at least a truth assignment satisfies all clauses, hence the formula is satisfiable. The signal object `yes` is produced and sent out of the system. The computation halts. (If the formula is not satisfiable, it halts after $m+1$ steps.)

Therefore, if the system halts without sending out the object `yes`, then the formula was unsatisfiable; if the system sends out the object `yes`, in step $m+3$, then the formula is satisfiable.

Note that the initial configuration of the system, the rules of each compartment included, is of a polynomial size with respect to n and m; the total alphabet of the system is exponential in size, but the objects are constructed during the computation, not while constructing the system itself.

The construction is uniform (it starts from the problem itself, not from a given instance of the problem – in membrane computing, the weaker case when we start from an instance of the decision problem, is called semi-uniform), and this concludes the proof. □

The previous system has been presented with two membranes, in order to make clear the ω component of it, the inner membrane, with a "standard" membrane around it. The construction can be easily modified in order to obtain a system with only one membrane (decreasing superscripts and subscripts can be done in the same way for objects supposed to appear in arbitrarily many copies, because each object is modified by only one rule and all copies of an object evolve at the same time, in the same way). Thus, the previous theorem can be also considered as a fypercomputation result for R systems instead of P systems.

We do not know whether the previous result can be improved by imposing the restriction to use only non-cooperative rules.

6 Final Remarks

Following the model of hypercomputation, we have introduced the notion and we considered the issue of "fypercomputation" – of speeding-up computations (of P systems) to such an extent to obtain polynomial time solutions to intractable (typically, **NP**-complete) problems. After briefly mentioning the existing speed-up ideas investigated in membrane computing (membrane division, membrane creation, string replication, and using pre-computed resources), we propose further ideas, such as (local) acceleration, two-dimensional time, using arbitrary populations of objects (like in reaction systems, i.e., assuming for each object that it is either absent or present in arbitrarily many copies). An example of fypercomputation is given for the last idea.

The paper is preliminary and speculative, it mainly calls the attention to the systematic study of fypercomputations.

Acknowledgements. Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200. Detailed comments by two anonymous referees are gratefully acknowledged.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 226, 1021–1024 (1994)
2. Arulanandham, J.J.: Implementing Bead-Sort with P Systems. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) UMC 2002. LNCS, vol. 2509, pp. 115–125. Springer, Heidelberg (2002)
3. Arulanandham, J.J., Calude, C.S., Dinneen, M.J.: Balance Machines: Computing = Balancing. In: Jonoska, N., Păun, G., Rozenberg, G. (eds.) Aspects of Molecular Computing. LNCS, vol. 2950, pp. 36–48. Springer, Heidelberg (2003)
4. Calude, C.S., Păun, G.: Bio-steps beyond Turing. BioSystems 77, 175–194 (2004)
5. Castellanos, J., Păun, G., Rodriguez-Patón, A.: Computing with membranes: P systems with worm-objects. In: Proc. IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000, La Coruna, Spain, pp. 64–74 (2000)
6. Copeland, J.: Hypercomputation. Minds and Machines 12, 461–502 (2002)
7. Dassow, J., Păun, G.: On the power of membrane computing. J. of Universal Computer Science 5, 33–49 (1999)
8. Davis, M.: Why there is no such discipline as hypercomputation. Applied Mathematics and Computation (Special Issue on Hypercomputation) 178, 4–7 (2006)
9. Eberbach, E., Wegner, P.: Beyond Turing machines. Bulletin of the EATCS 81, 279–304 (2003)
10. Ehrenfeucht, A., Rozenberg, G.: Basic Notions of Reaction Systems. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 27–29. Springer, Heidelberg (2004)
11. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae 75, 263–280 (2007)
12. Ehrenfeucht, A., Rozenberg, G.: Introducing time in reaction systems. Theoretical Computer Sci. 410, 310–322 (2009)
13. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71, 279–308 (2006)
14. Ishdorj, T.-O., Leporati, A.: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. Natural Computing 7, 519–534 (2008)
15. Leporati, A., Gutiérrez-Naranjo, M.A.: Solving SUBSET-SUM by spiking neural P systems with pre-computed resources. Fundamenta Informaticae 87, 61–77 (2008)
16. Martin-Vide, C., Păun, G., Rodriguez-Patón, A.: On P systems with membrane creation. Computer Science J. of Moldova 9, 134–145 (2001)
17. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000) (first circulated as Turku Center for Computer Science-TUCS Report 208 (November 1998), `www.tucs.fi`
18. Păun, G.: P systems with active membranes: Attacking NP-complete problems. J. Automata, Languages, and Combinatorics 6, 75–90 (2001)
19. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
20. Păun, G., Pérez-Jiménez, M.J.: Towards bridging two cell-inspired models: P systems and R systems. Theoretical Computer Sci. (to appear)
21. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
22. Pérez-Jímenez, M.J.: An Approach to Computational Complexity in Membrane Computing. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 85–109. Springer, Heidelberg (2005)

23. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity – Membrane Division, Membrane Creation. In: [21], ch. 12, pp. 302–336
24. Putz, V., Svozil, K.: Can a computer be "pushed" to perform faster-than-light? In: Proc. UC 2010 Hypercomputation Workshop "HyperNet 2010", Univ. of Tokyo, June 22 (2010)
25. Siegelmann, H.: Computation beyond the Turing limit. Science 238, 632–637 (1995)
26. Sosík, P., Valík, O.: On Evolutionary Lineages of Membrane Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 67–78. Springer, Heidelberg (2006)
27. Syropoulos, A.: Hypercomputation: Computing Beyond the Church-Turing Barrier. Springer, Berlin (2008)
28. van Leeuwen, J., Wiedermann, J.: Beyond the Turing Limit: Evolving Interactive Systems. In: Pacholski, L., Ružička, P. (eds.) SOFSEM 2001. LNCS, vol. 2234, pp. 90–109. Springer, Heidelberg (2001)
29. Zandron, C.: A Model for Molecular Computing: Membrane Systems. PhD thesis, Milano Univ., Italy (2002)
30. The P Systems Website, `http://ppage.psystems.eu`