The P Versus NP Problem Through Cellular Computing with Membranes

Mario J. Pérez-Jiménez, Alvaro Romero-Jiménez, and
Fernando Sancho-Caparrini

Abstract. We study the **P** *versus* **NP** problem through membrane systems. Language accepting P systems are introduced as a framework allowing us to obtain a characterization of the $\mathbf{P} \neq \mathbf{NP}$ relation by the polynomial time unsolvability of an **NP**–complete problem by means of a P system.

1 Introduction

The **P** *versus* **NP** problem [2] is the problem of determining whether every language accepted by some non-deterministic algorithm in polynomial time is also accepted by some deterministic algorithm in polynomial time. To define the above problem precisely we must have a formal definition for the concept of an algorithm. The theoretical model to be used as a computing machine in this work is the Turing machine, introduced by Alan Turing in 1936 [10], several years before the invention of modern computers.

A deterministic Turing machine has a transition function providing a functional relation between configurations; so, for every input there exists only one computation (finite or infinite), allowing us to define in a natural way when an input is *accepted* (through an accepting computation).

In a non-deterministic Turing machine, for a given configuration several successor configurations can exist. Therefore, it could happen that for a given input different computations exist. In these machines, an input is accepted if there exists at least one finite accepting computation associated with it.

The class **P** is the class of languages accepted by some deterministic Turing machine in a time bounded by a polynomial on the length (size) of the input. From an informal point of view, the languages in the class **P** are identified with the problems having an efficient algorithm that gives an answer in a feasible time; the problems in **P** are also known as *tractable* problems.

The class **NP** is the class of languages accepted by some non-deterministic Turing machine where for every accepted input there exists at least one accepting computation taking an amount of steps bounded by a polynomial on the length of the input.

Every deterministic Turing machine can be considered as a non-deterministic one, so we have $\mathbf{P} \subseteq \mathbf{NP}$. In terms of the previously defined classes, the **P** *versus* **NP** problem can be expressed as follows: is it verified the relation $\mathbf{NP} \subseteq \mathbf{P}$?

The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question is one of the outstanding open problems in theoretical computer science. The relevance of this question does not lie only in the inherent pleasure of solving a mathematical problem, but in this case an answer to it could provide an information of a high practical interest. For instance, a negative answer to this question would confirm that the majority of current cryptographic systems are secure from a practical point of view. On the other hand, a positive answer could not only entail the vulnerability of cryptographic systems, but this kind of answer is expected to come together with a general procedure which will provide a deterministic algorithm solving any **NP**-complete problem in polynomial time.

Moreover, the problems known to be in the class **NP** but not known to be in **P** are varied and of highest practical interest. An **NP**–complete problem is a *hardest* (in certain sense) problem in **NP**; that is, any problem in **NP** could be efficiently solved using an efficient algorithm which solves a fixed **NP**–complete problem. These problems are the suitable candidates to attack the **P** *versus* **NP** problem.

In the last years several computing models using powerful and inherent tools inspired from nature have been developed (because of this reason, they are known as *bio-inspired* models) and several solutions in polynomial time to problems from the class **NP** have been presented, making use of non-determinism or of an exponential amount of space. This is the reason why a practical implementation of such models (in biological, electronic, or other media) could provide a quantitative improvement for the resolution of **NP**-complete problems.

In this work we focus on one of these models, the *cellular computing model with membranes*, specifically, on one of its variants, the language accepting P systems, in order to develop a computational complexity theory allowing us to attack the **P** *versus* **NP** problem from other point of view than the classical one.

The paper is structured as follows. The next section is devoted to the definition of language accepting P systems. In section 3 a polynomial complexity class for the above model is introduced. Sections 4 and 5 provides simulations of deterministic Turing machines by P systems and language accepting P systems by deterministic Turing machines. Finally, in section 6 we establish a characterization of the **P** *versus* **NP** problem through P systems.

2 Language Accepting P Systems

Until the end of 90's decade several natural computing models have been introduced simulating the way nature *computes* at the genetic level (genetic algorithms and DNA based molecular computing) and at the neural level (neural networks). In 1998, Gh. Păun [5] suggests a new level of computation: *the cellular level*.

Cells can be considered as machines performing certain computing processes; in the distributed framework of the hierarchical arrangement of internal vesicles, the communication and alteration of the chemical components of the cell are carried out. Of course, the processes taking place in the cell are complex enough for not attempting to completely model them. The goal is to create an abstract cell-like computing model allowing to obtain alternative solutions to problems which are intractable from a classical point of view.

The first characteristic to point out from the internal structure of the cell is the fact that the different units composing the cell are delimited by several types of *membranes* (in a broad sense): from the membrane that separates the cell from the environment into which the cell is placed, to those delimiting the inner vesicles. Also, with regard to the functionality of these membranes in nature, it has to be emphasized the fact that they do not generate isolated compartments, but they allow the chemical compounds to flow between them, sometimes in selective forms and even in only one direction. Similar ideas were previously considered, for instance, in [1] and [3].

P systems are described in [4] as follows: a *membrane structure* consists of several membranes arranged in a hierarchical structure inside a main membrane (called the *skin*) and delimiting *regions* (each region is bounded by a membrane and the immediately lower membranes, if there are any). Regions contain *multisets* of *objects*, that is, sets of objects with multiplicities associated with the elements. The objects are represented by symbols from a given alphabet. They evolve according to given *evolution rules*, which are also associated with the regions. The rules are applied non-deterministically, in a maximally parallel manner (in each step, all objects which *can* evolve *must* do so). The objects can also be moved (*communicated*) between regions. In this way, we get *transitions* from one *configuration* of the system to the next one. This process is synchronized: a global clock is assumed, marking the time units common to all compartments of the system. A sequence (finite or infinite) of transitions between configurations constitutes a *computation*; a computation which reaches a configuration where no rule is applicable to the existing objects is a *halting computation*. With each halting computation we associate a *result*, by taking into consideration the objects collected in a specified *output membrane* or in the environment.

For an exhaustive overview of transition P systems and of their variants and properties, see [4].

Throughout this paper, we will study the capacity of cellular systems with membranes to attack the efficient solvability of presumably intractable decision problems. We will focus on a specific variant of transition P systems: *language accepting P systems*. These systems have an *input membrane*, and work in such a way that when introducing in the input membrane a properly encoded string, a "message" is sent to the environment, encoding whether this string belongs or not to a specified language.

Definition 1. *A* membrane structure *is a rooted tree, where the nodes are called* membranes*, the root is called* skin*, and the leaves are called* elementary membranes*.*

Definition 2. *Let $\mu = (V(\mu), E(\mu))$ be a membrane structure. The* membrane structure with external environment *associated with μ is the rooted tree such that: (a) the root of the tree is a new node that we denote by env; (b) the set of nodes is $V(\mu) \cup \{env\}$; and (c) the set of edges is $E(\mu) \cup \{\{env, skin\}\}$.*

The node *env* is called *environment* of the structure μ. So, every membrane structure has associated in a natural way an environment.

Definition 3. *A* language accepting P system *(with input membrane and external output) is a tuple*

$$\Pi = (\Sigma, \Gamma, \Lambda, \#, \mu_\Pi, \mathcal{M}_1, ..., \mathcal{M}_p, (R_1, \rho_1), ..., (R_p, \rho_p), i_\Pi)$$

verifying the following properties:

- *The input alphabet of Π is Σ.*
- *The working alphabet of Π is Γ, with $\Sigma \subsetneq \Gamma$ and $\# \in \Gamma - \Sigma$.*
- *μ_Π is a membrane structure consisting of p membranes, with the membranes (and hence the regions) injectively labelled with $1, 2, \ldots, p$.*
- *i_Π is the label of the input membrane.*
- *The output alphabet of Π is $\Lambda = \{Yes, No\}$.*
- *$\mathcal{M}_1, ..., \mathcal{M}_p$ are multisets over $\Gamma - \Sigma$, representing the initial contents of the regions of $1, 2, \ldots, p$ of μ_Π.*
- *$R_1, ..., R_p$ are finite sets of evolution rules over Γ associated with the regions $1, 2, \ldots, p$ of μ_Π.*
- *ρ_i, $1 \leq i \leq p$, are partial order relations over R_i specifying a priority relation among rules of R_i.*

An *evolution rule* is a pair (u, v), usually represented $u \to v$, where u is a string over Γ and $v = v'$ or $v = v'\delta$, with v' a string over

$$\Gamma \times \big(\{here, out\} \cup \{in_i \mid i = 1, \ldots, p\}\big).$$

Consider a rule $u \to v$ from a set R_i. To apply this rule in membrane i means to remove the multiset of objects specified by u from membrane i (the latter must contain, therefore, sufficient objects so that the rule can be applied), and to introduce the objects specified by v, in the membranes indicated by the *target commands* associated with the objects from v.

Specifically, for each $(a, out) \in v$ an object a will exit the membrane i and will become an element of the membrane immediately outside it (that is, the father membrane of membrane i), or will leave the system and will go to the environment if the membrane i is the skin membrane. If v contains a pair $(a, here)$, then the object a will remain in the same membrane i where the rule is applied

(when specifying rules, pairs $(a, here)$ are simply written a, the indication *here* is omitted). For each $(a, in_j) \in v$ an object a should be moved in the membrane with label j, providing that this membrane is immediately inside membrane i (that is, membrane i is the father of membrane j); if membrane j is not directly accesible from membrane i (that is, if membrane j is not a child membrane of membrane i), then the rule cannot be applied. Finally, if δ appears in v, then membrane i is dissolved; that is, membrane i is removed from the membrane structure, and all objects and membranes previously present in it become elements of the immediately upper membrane (the father membrane) while the evolution rules and the priority relations of the dissolved membrane are removed. The skin membrane is never dissolved; that is, no rule of the form $u \to v'\delta$ is applicable in the skin membrane.

All these operations are done in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the membrane associated with the rules, and for all membranes at the same time.

The rules from the set R_i, $1 \leq i \leq p$, are applied to objects from membrane i synchronously, in a non-deterministic maximally parallel manner; that is, we assign objects to rules, non-deterministically choosing the rules and the objects assigned to each rule, but in such a way that after this assignation no further rule can be applied to the remaining objects. Therefore, a rule can be applied in the same step as many times as the number of copies of objects allows it.

On the other hand, we interpret the priority relations between the rules in a *strong sense*: a rule $u \to v$ in a set R_i can be used only if no rule of a higher priority exists in R_i and can be applied at the same time with $u \to v$.

A *configuration* of Π is a tuple $(\mu, M_E, M_{i_1}, \ldots, M_{i_q})$, where μ is a membrane structure obtained by removing from μ_Π all membranes different from $i_1, \ldots, i_q$ (of course, the skin membrane cannot be removed), M_E is the multiset of objects contained in the environment of μ, and M_{i_j} is the multiset of objects contained in the region i_j.

For every multiset m over Σ (the input alphabet of the P system), the *initial configuration of Π with input m* is the tuple $(\mu_\Pi, \emptyset, \mathcal{M}_1, ..., \mathcal{M}_{i_\Pi} \cup m, ..., \mathcal{M}_p)$. That is, in any initial configuration of Π the environment is empty. We will denote by I_Π the collection of possible inputs for the system Π.

Given a configuration C of a P system Π, applying properly the evolution rules as described above, we obtain, in a non-deterministic way, a new configuration C'. We denote by $C \Rightarrow_\Pi C'$, and we say that we have a *transition* from C to C'. A *halting configuration* is a configuration in which no evolution rule can be applied.

A *computation* $\mathcal{C}$ of a P system is a sequence of configurations, $\{C^i\}_{i<r}$, where: C^0 is an initial configuration of the system; $C^i \Rightarrow_\Pi C^{i+1}$, for every $i < r$; and, either $r \in \mathbb{N}^+$ (that is, it is a non-zero natural number) and C^{r-1} is a *halting configuration*, or $r = \infty$, in which case it is said that $\mathcal{C}$ is not halting.

For a computation $\mathcal{C} = \{C^i\}_{i<r}$ we will denote by M_E^j the content of the environment in the configuration C^j. Next we define the output of the P system.

Definition 4. *The output of a computation* $\mathcal{C} = \{C^i\}_{i<r}$ *is:*

$$Output(\mathcal{C}) = \begin{cases} Yes, & \text{if } \mathcal{C} \text{ is halting, } Yes \in M_E^{r-1} \text{ and } No \notin M_E^{r-1}, \\ No, & \text{if } \mathcal{C} \text{ is halting, } No \in M_E^{r-1} \text{ and } Yes \notin M_E^{r-1}, \\ not\ defined, & otherwise. \end{cases}$$

If $\mathcal{C}$ *satisfies any of the two first conditions, then we say that it is a* successful computation.

Definition 5. *A language accepting P system is said to be* valid *if every halting computation is a successful computation and every halting computation, and only them, sends out the symbol # (and only in the last step).*

We denote by $\mathcal{LA}$ the class of valid language accepting P systems.

Next we define what it means that such P systems *accept* or *decide* a language.

Definition 6. *Let* L *be a language over an alphabet* Ω*. We say that the system* $\Pi \in \mathcal{LA}$ accepts *the language* L *if the following properties are verified:*

- *There exists a total function,* $cod : \Omega^* \to I_\Pi$*, computable and injective, encoding strings over* Ω *by means of multisets over the input alphabet of* Π*.*
- *For every string* $w \in \Omega^*$ *it is verified that:*
  - *If* $w \in L$*, then there exists* a *computation* $\mathcal{C}$ *of* Π *with input* $cod(w)$ *such that* $\mathcal{C}$ *is halting and* $Output(\mathcal{C}) = Yes$*.*
  - *If there exists* a *computation* $\mathcal{C}$ *of* Π *with input* $cod(w)$ *such that* $\mathcal{C}$ *is halting and* $Output(\mathcal{C}) = Yes$*, then* $w \in L$*.*

Definition 7. *Let* L *be a language over an alphabet* Ω*. We say that the system* $\Pi \in \mathcal{LA}$ decides *the language* L *if the following properties are verified:*

- *Every computation of* Π *is halting.*
- *There exists a total function,* $cod : \Omega^* \to I_\Pi$*, computable and injective, encoding strings over* Ω *by means of multisets over the input alphabet of* Π*.*
- *For every string* $w \in \Omega^*$ *it is verified that:*
  - *If* $w \in L$*, then for* every *computation* $\mathcal{C}$ *of* Π *with input* $cod(w)$ *it is verified that* $Output(\mathcal{C}) = Yes$*.*
  - *If* $w \notin L$*, then for* every *computation* $\mathcal{C}$ *of* Π *with input* $cod(w)$ *it is verified that* $Output(\mathcal{C}) = No$*.*

3 A Polynomial Complexity Class in Cellular Systems

In order to give a formal definition of computational complexity classes in this model, we have to first specify what we mean by a decision problem.

Definition 8. *A* decision problem*, X, is a pair (I_X, θ_X) such that I_X is a language (over a finite alphabet) whose elements are called* instances *of the problem and θ_X is a total Boolean function over I_X.*

A decision problem X is solvable by a Turing machine TM if I_X is the set of inputs of TM, for any $w \in I_X$ the Turing machine halts over w, and w is accepted if and only if $\theta_X(w) = 1$.

To solve a problem by means of P systems, we usually construct a family of such devices so that each element decides the instances of *equivalent size*, in a certain sense which will be specified below.

Definition 9. *Let $g : \mathbf{N}^+ \to \mathbf{N}^+$ be a total computable function. We say that a decision problem X is solvable by a family of valid language accepting P systems, in a time bounded by g, and we denote this by $X \in MC_{\mathcal{LA}}(g)$, if there exists a family of P systems, $\mathbf{\Pi} = \big(\Pi(n)\big)_{n\in\mathbb{N}^+}$, with the following properties:*

1. *For every $n \in \mathbb{N}$ it is verified that $\Pi(n) \in \mathcal{LA}$.*
2. *There exists a Turing machine constructing $\Pi(n)$ from n in polynomial time (we say that Π is polynomially uniform by Turing machines).*
3. *There exist two functions, $cod : I_X \to \bigcup_{n\in\mathbb{N}^+} I_{\Pi(n)}$ and $s : I_X \to \mathbb{N}^+$, computable in polynomial time, such that:*
   - *For every $w \in I_X$, $cod(w) \in I_{\Pi(s(w))}$.*
   - *The family $\mathbf{\Pi}$ is bounded, with regard to (X, cod, s, g); that is, for each $w \in I_X$ every computation of the system $\Pi(s(w))$ with input $cod(w)$ is halting and, moreover, it performs at most $g(|w|)$ steps.*
   - *The family $\mathbf{\Pi}$ is sound, with regard to (X, cod, s); that is, for each $w \in I_X$ if there exists an accepting computation of the system $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.*
   - *The family $\mathbf{\Pi}$ is complete, with regard to (X, cod, s); that is, for each $w \in I_X$ if $\theta_X(w) = 1$, then every computation of the system $\Pi(s(w))$ with input $cod(w)$ is an accepting computation.*

Note that we impose a certain kind of *confluence* of the systems, in the sense that every computation with the same input must return the same output.

As usual, the polynomial complexity class is obtained using as bounds the polynomial functions.

Definition 10. *The class of decision problems solvable in polynomial time by a family of cellular computing systems belonging to the class $\mathcal{LA}$, is*

$$\mathbf{PMC}_{\mathcal{LA}} = \bigcup_{g\ poly.} MC_{\mathcal{LA}}(g).$$

This complexity class is closed under polynomial-time reducibility.

Proposition 1. *Let X and Y be two decision problems such that X is polynomial-time reducible to Y. If $Y \in \mathbf{PMC}_{\mathcal{LA}}$, then $X \in \mathbf{PMC}_{\mathcal{LA}}$.*

4 Simulating Deterministic Turing Machines by P Systems

In this section we consider deterministic Turing machines as language decision devices. That is, the machines halt over any string on the input alphabet, with the halting state equal to the accepting state, in the case that the string belongs to the decided language, and with the halting state equal to the rejecting state in the case that the string does not belong to the language.

It is possible to associate with a Turing machine a decision problem, and this will permit us to define what means that such a machine is simulated by a family of P systems.

Definition 11. *Let TM be a Turing machine with input alphabet Σ_{TM}. The* decision problem associated with TM *is the problem $X_{TM} = (I, \theta)$, where $I = \Sigma^*_{TM}$, and for every $w \in \Sigma^*_{TM}$, $\theta(w) = 1$ if and only if TM accepts w.*

Obviously, the decision problem X_{TM} is solvable by the Turing machine TM.

Definition 12. *We say that a Turing machine TM is simulated in polynomial time by a family of systems of the class $\mathcal{LA}$, if $X_{TM} \in \mathbf{PMC}_{\mathcal{LA}}$.*

Next we state that every deterministic Turing machine can be simulated in polynomial time by a family of systems of the class $\mathcal{LA}$.

Proposition 2. *Let TM be a deterministic Turing machine working in polynomial time. Then $X_{TM} \in \mathbf{PMC}_{\mathcal{LA}}$.*

See chapter 9 of [8], which follows ideas from [9], for details of the proof.

5 Simulating Language Accepting P Systems by Deterministic Turing Machines

In this section we are going to prove that if a decision problem can be solved in polynomial time by a family of language accepting P systems, then it can also be solved in polynomial time by a deterministic Turing machine.

For the design of the Turing machine we were inspired by the work of C. Zandron, C. Ferretti and G. Mauri [11], with the difference that the mentioned paper deals with P systems with active membranes.

Proposition 3. *For every decision problem solvable in polynomial time by a family of valid language accepting P systems, there exists a Turing machine solving the problem in polynomial time.*

Proof. Let X be a decision problem such that $X \in \mathbf{PMC}_{\mathcal{LA}}$. Then, there exists a family of valid language accepting P systems $\mathbf{\Pi} = \big(\Pi(n)\big)_{n \in \mathbb{N}^+}$ such that:

1. The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.
2. There exist two functions $cod : I_X \to \bigcup_{n\in\mathbb{N}^+} I_{\Pi(n)}$ and $s : I_X \to \mathbb{N}^+$, computable in polynomial time, such that:
   - For every $w \in I_X$, $cod(w) \in I_{\Pi(s(w))}$.
   - The family $\mathbf{\Pi}$ is polynomially bounded, with regard to (X, cod, s).
   - The family $\mathbf{\Pi}$ is sound and complete, with regard to (X, cod, s).

Given $n \in \mathbb{N}^+$, let A_n be the number of symbols in the input alphabet of $\Pi(n)$, B_n the number of symbols in the working alphabet, C_n the number of symbols in the output alphabet, D_n the number of membranes, E_n the maximum size of the multisets initially associated with them, F_n the total number of rules of the system, and G_n the maximum length of them. Since the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, these numbers are polynomial with respect to n.

Let m be an input multiset of the system $\Pi(n)$. Given a computation $\mathcal{C}$ of $\Pi(n)$ with input m, we denote by $H_n(m)$ the maximum number of digits, in base 2, of the multiplicities of the objects contained in the multisets associated with the membranes of the systems and with the environment, in any step of $\mathcal{C}$. Naturally, this number depends on $\mathcal{C}$, but what we are interested in, and we will prove at the end of the proof, is that *any* computation of the system $\Pi(s(w))$ with input $cod(w)$ verifies that $H_{s(w)}(cod(w))$ is polynomial in the size of the string w.

Next, we associate with the system $\Pi(n)$ a deterministic Turing machine, $TM(n)$, *with multiple tapes*, such that, given an input multiset m of $\Pi(n)$, the machine reproduces *a* specific computation of $\Pi(n)$ over m.

The input alphabet of the machine $TM(n)$ coincides with that of the system $\Pi(n)$. On the other hand, the working alphabet contains, besides the symbols of the input alphabet of $\Pi(n)$ the following symbols: a symbol for each label assigned to the membranes of $\Pi(n)$; the symbols 0 and 1, that will allow to operate with numbers represented in base 2; three symbols indicating if a membrane has not been dissolved, has to be dissolved or has been dissolved; and three symbols that will indicate if a rule is awaiting, is applicable or is not applicable.

Subsequently, we specify the tapes of this machine.

- We have one *input tape*, that keeps a string representing the input multiset received.
- For each membrane of the system we have:
  - One *structure tape*, that keeps in the second cell the label of the father membrane, and in the third cell one of the three symbols that indicate if the membrane has not been dissolved, if the membrane has to dissolve, or if the membrane has been dissolved.
  - For each object of the working alphabet of the system:
    - One *main tape*, that keeps the multiplicity of the object, in base 2, in the multiset contained in the membrane.
    - One *auxiliary tape*, that keeps temporary results, also in base 2, of applying the rules associated with the membrane.

  - One *rules tape*, in which each cell starting with the second one corresponds to a rule associated with the membrane (we suppose that the set of those rules is ordered), and keeps one of the three symbols that indicate whether the rule is awaiting, it is applicable, or it is not applicable.
- For each object of the output alphabet we have:
  - One *environment tape*, that keeps the multiplicity of the object, in base 2, in the multiset associated with the environment.

Next we describe the steps performed by the Turing machine in order to simulate the P system. Take into account that, making a breadth first search traversal (with the skin as source) on the *initial* membrane structure of the system $\Pi(n)$, we obtain a natural order between the membranes of $\Pi(n)$. In the algorithms that we specify below we consider that they always traverse *all* the membranes of the *original* membrane structure and that, moreover, they do it in the order induced by the breadth traversal of that structure.

I. Initialization of the system. In the first phase of the simulation process followed by the Turing machine the symbols needed to reflect the initial configuration of the computation with input m that is going to be simulated are included in the corresponding tapes.

for all membrane *mb* of the system **do**
  if *mb* is not the skin membrane **then**
    – Write in the second cell of the structure tape of *mb* the label corresponding to the father of *mb*
  end if
  – Mark *mb* as non-dissolved membrane in the third cell of the structure tape of *mb*
  for all symbol *ob* of the working alphabet **do**
    – Write in the main tape of *mb* for *ob* the multiplicity, in base 2, of *ob* in the multiset initially associated with *mb*
  end for
end for
for all symbol *ob* of the input alphabet **do**
  – Read the multiplicity, in base 2, of *ob* in the input tape
  – Add that multiplicity to the main tape of the input membrane for *ob*
end for

II. Determine the applicable rules. To simulate a step of the cellular computing system, what the machine has to do first is to determine the set of rules that are applicable (each of them independently) to the configuration considered in the membranes they are associated with.

for all membrane *mb* of the system **do**
 if *mb* has not been dissolved **then**
   for all rule *r* associated with *mb* **do**
     – Mark *r* as awaiting rule
   end for

for all rule r associated with mb **do**
if – r is awaiting and
– mb contains the antecedent of r and
– r only sends objects to child membranes of mb that have not been dissolved and
– r does not try to dissolve the skin membrane
then
– Mark r as applicable rule
for all rule r' associated with mb of lower priority than r **do**
– Mark r' as non-applicable rule
end for
else
– Mark r as non-applicable rule
end if
end for
end if
end for

III. Apply the rules. Once the applicable rules are determined, they are applied in a maximal manner to the membranes they are associated with. The fact that the rules are considered in a certain order (using local maximality for each rule, according to that order) determines a specific applicable multiset of rules, thus fixing the computation of the system that the Turing machine simulates. However, from Definition 9 of complexity class it will follow that the chosen computation is not relevant for the proof, due to the *confluence* of the system.

for all membrane mb of the system **do**
if mb has not been dissolved **then**
for all rule r associated with mb that is applicable **do**
for all object ob in the antecedent of r **do**
–Compute the integer quotient that results from dividing the multiplicity of ob in the main tape of mb by the multiplicity of ob in the antecedent of r
end for
– Compute the minimum of the values obtained in the previous loop (that minimum is the maximum number of times that the rule r can be applied to membrane mb). Let us call it *index* of the rule r.
for all object ob in the antecedent of r **do**
– Multiply the multiplicity of ob in the antecedent of r by the index of r
– Erase the result obtained from the main tape of mb for the object ob
end for
for all object ob in the consequent of r **do**
– Multiply the multiplicity of ob in the consequent of r by the index of r

– Add the result obtained to the auxiliary tape for ob in the corresponding membrane

end for
if r dissolves mb **then**
– Mark mb as membrane to dissolve in the third cell of the structure tape of mb
end if
end for
end if
end for

IV. Update the multisets. After applying the rules, the auxiliary tapes keep the results obtained, and then these results have to be moved to the corresponding main tapes.

for all membrane mb of the system **do**
if mb has not been dissolved **then**
– Copy the content of the auxiliary tapes of mb into the corresponding main tapes
end if
end for

V. Dissolve the membranes. To finish the simulation of one step of the computation of the P system it is necessary to dissolve the membranes according to the rules that have been applied in the previous phase and to rearrange accordingly the structure of membranes.

for all membrane mb of the system **do**
if – mb has not been dissolved and
– the father of mb is marked as membrane to dissolve
then
– Make the father of mb equal to the father of the father of mb
end if
end for
for all membrane mb of the system **do**
if mb is marked as membrane to dissolve **then**
– Copy the contents of the main tapes of mb into the main tapes of the (possibly new) father of mb
– Mark mb as dissolved membrane in the third cell of the structure tape of mb
end if
end for

VI. Check if the simulation has ended. Finally, after finishing the simulation of one transition step of the computation of $\Pi(n)$, the Turing machine has to check if a halting configuration has been reached and, in that case, if the computation is an accepting or a rejecting one.

if the environment tape contains the symbol # **then**
if the environment tape contains the symbol Yes **then**
– Halt and accept the multiset m
else
– Halt and reject the multiset m
end if
else
– Simulate again a step of the computation of the P system
end if

It is easy to check that the family $\big(TM(n)\big)_{n\in\mathbb{N}^+}$ can be constructed in an uniform way and in polynomial time from $n \in \mathbb{N}^+$.

Let us finally consider the deterministic Turing machine $TM_{\mathbf{\Pi}}$ that works as follows:

Input: $w \in I_X$

– Compute $s(w)$
– Construct $TM(s(w))$
– Compute $cod(w)$
– Simulate the functioning of $TM(s(w))$ with input $cod(w)$

Then, the following assertions are verified:

1. The machine $TM_{\mathbf{\Pi}}$ works in polynomial time over $|w|$.
   Since the functions cod and s are polynomial in $|w|$, the numbers $A_{s(w)}$, $B_{s(w)}$, $C_{s(w)}$, $D_{s(w)}$, $E_{s(w)}$, $F_{s(w)}$, and $G_{s(w)}$ are polynomial in $|w|$.
   On the other hand, the family $\mathbf{\Pi}$ is polynomially bounded, with regard to (X, cod, s). Therefore, every computation of the system $\Pi(s(w))$ with input $cod(w)$ performs a polynomial number of steps on $|w|$. Consequently, the number of steps, P_w, performed by the computation simulated by the machine $TM(s(w))$ over $cod(w)$ is polynomial in $|w|$.
   Hence, the maximal multiplicity of the objects contained in the multisets associated with the membranes is in the order of $O(E_{s(w)} \cdot G_{s(w)}^{P_w})$. This implies that $H_{s(w)}(cod(w))$ is in the order of $O(P_w \cdot log_2(E_{s(w)} \cdot G_{s(w)}))$; that is, polynomial in $|w|$.
   It follows that the total time spent by $TM_{\mathbf{\Pi}}$ when receiving w as input is polynomial in $|w|$.
2. Let us suppose that $TM_{\mathbf{\Pi}}$ accepts the string w. Then the computation of $\Pi(s(w))$ with input $cod(w)$ simulated by $TM(s(w))$ is an accepting computation. Therefore $\theta_X(w) = 1$.
3. Let us suppose that $\theta_X(w) = 1$. Then every computation of $\Pi(s(w))$ with input $cod(w)$ is an accepting computation. Therefore, it is also the computation simulated by $TM(s(w))$. Hence $TM_{\mathbf{\Pi}}$ accepts the string w.

Consequently, we have proved that $TM_{\mathbf{\Pi}}$ solves X in polynomial time.

6 Characterizing the P ≠ NP Relation Through P Systems

Next, we establish characterizations of the $\mathbf{P} \neq \mathbf{NP}$ relation by means of the polynomial time unsolvability of **NP**–complete problems by families of language accepting P systems.

Theorem 1. *The following propositions are equivalent:*

1. $\mathbf{P} \neq \mathbf{NP}$.
2. $\exists X\big(X \text{ is an } \mathbf{NP}\text{–complete decision problem } \wedge X \notin \mathbf{PMC}_{\mathcal{LA}}\big)$.
3. $\forall X\big(X \text{ is an } \mathbf{NP}\text{–complete decision problem } \rightarrow X \notin \mathbf{PMC}_{\mathcal{LA}}\big)$.

Proof. To prove the implication $1 \Rightarrow 3$, let us suppose that there exists an **NP**–complete problem X such that $X \in \mathbf{PMC}_{\mathcal{LA}}$. Then, from Proposition 3 there exists a deterministic Turing machine solving the problem X in polynomial time. Hence, $X \in \mathbf{P}$. Therefore, $\mathbf{P} = \mathbf{NP}$, which leads to a contradiction.

The implication $3 \Rightarrow 2$ is trivial, because the class of **NP**–complete problems is non empty.

Finally, to prove the implication $2 \Rightarrow 1$, let X be an **NP**–complete problem such that $X \notin \mathbf{PMC}_{\mathcal{LA}}$. Let us suppose that $\mathbf{P} = \mathbf{NP}$. Then $X \in \mathbf{P}$. Therefore there exists a deterministic Turing machine TM that solves the problem X in polynomial time.

By Proposition 2, the problem X_{TM} is in $\mathbf{PMC}_{\mathcal{LA}}$. Then there exists a family $\mathbf{\Pi}_{TM} = \big(\Pi_{TM}(k)\big)_{k\in\mathbb{N}^+}$ of valid language accepting P systems simulating TM in polynomial time (with associated functions cod_{TM} and s_{TM}).

We consider the function $cod_X : I_X \rightarrow \bigcup_{k\in\mathbb{N}^+} I_{\Pi_{TM}(k)}$, given by $cod_X(w) = cod_{TM}(w)$, and the function $s_X : I_X \rightarrow \mathbb{N}^+$, given by $s_X(w) = |w|$. Then:

- The family $\mathbf{\Pi}_{TM}$ is polynomially uniform by Turing machine, and polynomially bounded, with regard to (X, cod_X, s_X).
- The family $\mathbf{\Pi}_{TM}$ is sound, with regard to (X, cod_X, s_X). Indeed, let $w \in I_X$ be such that there exists a computation of the system $\Pi_{TM}(s_X(w)) = \Pi_{TM}(s_{TM}(w))$ with input $cod_X(w) = cod_{TM}(w)$ that is an accepting computation. Then $\theta_{TM}(w) = 1$. Therefore $\theta_X(w) = 1$.
- The family $\mathbf{\Pi}_{TM}$ is complete, with regard to (X, cod_X, s_X). Indeed, let $w \in I_X$ be such that $\theta_X(w) = 1$. Then TM accepts the string w. Therefore $\theta_{TM}(w) = 1$. Hence, every computation of the system $\Pi_{TM}(s_{TM}(w)) = \Pi_{TM}(s_X(w))$ with input $cod_{TM}(w) = cod_X(w)$ is an accepting computation.

Consequently, $X \in \mathbf{PMC}_{\mathcal{LA}}$, and this leads to a contradiction.

Acknowledgement. The authors wish to acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

1. Berry, G.; Boudol, G. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992, pp. 217–248.
2. Cook, S. The P versus NP problem. Manuscript prepared for the Clay Mathematics Institute for the Millennium Prize Problems (revised November, 2000).
3. Manca, V. String rewriting and metabolism: A logical perspective. In *Computing with Bio-Molecules. Theory and Experiments* (Gh. Păun, ed.), Springer-Verlag, Singapore, 1998, pp. 36–60.
4. Păun, G. *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
5. Păun G. Computing with membranes, *Journal of Computer and System Sciences*, 61 (1), 2000, pp. 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
6. Păun, G.; Rozenberg, G. A guide to membrane computing, *Theoretical Computer Science*, 287, 2002, pp. 73–100.
7. Pérez–Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F. *Teoría de la Complejidad en Modelos de Computación con Membranas*, Ed. Kronos, Sevilla, 2002.
8. Romero-Jiménez, A. *Complexity and Universality in Cellular Computing Models*, PhD. Thesis, University of Seville, Spain, 2003.
9. Romero-Jiménez, A.; Pérez Jiménez, M.J. Simulating Turing machines by P systems with external output. *Fundamenta Informaticae*, vol. 49 (1-3), 2002, pp. 273–287.
10. Turing, A. On computable numbers with an application to the Entscheidnungsproblem. *Proceeding London Mathematical Society*, serie 2, 42, 1936-7, pp. 230–265.
11. Zandron, C.; Ferreti, C.; Mauri, G. Solving NP-complete problems using P systems with active membranes. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou; C. Calude; M.J. Dinneen, eds.), Springer-Verlag, Berlin, 2000, pp. 289–301.