Decision P systems and the P≠NP conjecture

Mario J. PÉREZ JIMÉNEZ, Álvaro ROMERO JIMÉNEZ, and
Fernando SANCHO CAPARRINI

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla, España

{Mario.Perez,Alvaro.Romero,Fernando.Sancho}@cs.us.es

Abstract. In this paper we present the model of *decision* P systems with external output and prove the following main result: if there exists an NP–complete problem that cannot be solved in polynomial time, with regard to the input length, by the deterministic variant of such P systems, *constructed* in polynomial time, then $P \neq NP$. From Zandron-Ferreti-Mauri's theorem it follows that if $P \neq NP$ then no NP–complete problem can be solved in polynomial time, with regard to the input length, by a deterministic P system with active membranes but without membrane division, *constructed* in polynomial time from the input. Both results give a characterization of $P \neq NP$ through the solvability by deterministic P systems.

1 Introduction

In [2] a new model of computation, called *P Systems*, within the framework of *Natural Computing* (bio-inspired computing), is introduced by G. Păun. It is based upon the notion of *membrane structure* that is used to enclose *computing cells* in order to make them independent computing units. Also, a membrane serves as a communication channel between a given cell and other cells "adjacent" to it. This model comes from the observation that the processes which take place in the complex structure of a living cell can be considered as *computations*.

Since these computing devices were introduced several variants have been considered. A fairly complete compendium about P systems can be found at [7]. In particular, P systems with external output are studied in [4].

The different variants of P systems found in the literature are generally thought as generating devices. Many of them have been proved to be computationally complete: they compute all Turing computable sets of natural numbers or all recursively enumerable languages, depending on the variant considered.

The model we research here has two characteristics that have seldom been considered before: we work with *decision* devices with *input data*. The aim is that this kind of P systems allows us to deal with decision problems.

The main goal of this paper is to show a sufficient condition for $P \neq NP$: if there exists an NP–complete problem that cannot be solved in polynomial time, with regard to the input length, by deterministic decision P systems with external output, *constructed* in polynomial time, then $P \neq NP$.

To achieve it, we prove that every decision problem which can be solved by a deterministic Turing machine in polynomial time can also be solved by deterministic decision P systems in polynomial time.

The paper is organized as follows: Section 2 briefly presents some basic concepts about P systems with external output; Section 3 introduces the new model of *decision* P systems with external output; Section 4 studies how to simulate deterministic Turing machines by families of such P systems; Section 5 establishes our main results about decision P systems and the $P \neq NP$ conjecture.

2 Multisets. Membrane structures. Evolution rules

A *multiset* over a set, A, is an application $m : A \rightarrow \mathbb{N}$. A multiset is said to be empty (resp. finite) if its support, $supp(m) = \{a \in A : m(a) > 0\}$, is empty (resp. finite). If m is a finite multiset over A, we will denote it $m = \{\{a_1, \ldots, a_m\}\}$, where the elements $a_i \in supp(m)$ are possibly repeated. We write $M(A)$ for the set of all the multisets over A.

The set of *membrane structures*, MS, is defined by recursion as follows: 1. $[\,] \in MS$; 2. If $\mu_1, \ldots, \mu_n \in MS$, then $[\mu_1 \ldots \mu_n] \in MS$.

A membrane structure, μ, can also be seen as a rooted tree, $\big(V(\mu), E(\mu)\big)$. Then, the nodes of this tree are called *membranes*, the root node the *skin membrane* and the leaves *elementary membranes* of the membrane structure. The *degree* of a membrane structure is the number of membranes in it.

The *membrane structure with external environment* associated with a membrane structure, μ, is $\mu^E = [_E \mu]_E$. If we consider the latter as a rooted tree, the root node is called the *external environment* of μ.

Given an alphabet, Γ, we associate with every membrane of a membrane structure a finite multiset of elements of Γ, which are called the *objects* of the membrane.

We also associate with every one of these membranes a finite set of *evolution rules.* A evolution rule over Γ is a pair (u, v), usually written $u \rightarrow v$, where u is a string over Γ and $v = v'$ or $v = v'\delta$, where v' is a string over

$$\Gamma \times (\{here, out\} \cup \{in_l : l \in V(\mu)\})$$

and δ is a special symbol not in Γ. The idea behind a rule is that the objects in u "evolve" into the objects in v', moving or not to another membrane and possibly dissolving the original one.

The *length* of a rule is the number of symbols involved in the rule.

3 Decision P Systems with External Output

Definition 1. *A decision P system with external output is a tuple*

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

where

- *Σ is an alphabet, called the input alphabet.*
- *Γ is an alphabet such that $\Sigma \subseteq \Gamma$; its elements are called objects.*
- *μ_{Π} is a membrane structure of degree p, the membranes of which we suppose labeled from 1 to p.*
- *The input membrane of Π is labeled by $i_{\Pi} \in \{1, \dots, p\}$.*
- *$\mathcal{M}_i$ is a multiset over $\Gamma - \Sigma$ associated with the membrane labeled by i, for every $i = 1, \dots, p$.*
- *R_i is a finite set of evolution rules over Γ associated with the membrane labeled by i, for every $i = 1, \dots, p$.*
- *$YES, NO \in \Gamma - \Sigma$.*

To formalize the semantics of this model we define first what a configuration of such a P system is, from what follows the notion of computation.

Definition 2. *Let Π be a decision P system with external output.*

1. *A configuration of Π is a pair (μ^E, M), where μ is a membrane structure such that $V(\mu) \subseteq V(\mu_{\Pi})$ and it has the same root than μ_{Π}, and M is an application from $V(\mu^E)$ into $M(\Gamma)$. For every node $nd \in V(\mu^E)$ we denote $M_{nd} = M(nd)$.*
2. *The initial configuration of Π for the multiset $m \in M(\Sigma)$ is the pair (μ^E, M), where $\mu = \mu_{\Pi}$, $M_E = \emptyset$, $M_{i_{\Pi}} = m \cup \mathcal{M}_{i_{\Pi}}$ and $M_j = \mathcal{M}_j$, for every $j \neq i_{\Pi}$.*

The idea is that for every input multiset $m \in M(\Sigma)$, we add that multiset to the input membrane, i_{Π}, of the P system and then “start” this latter one.

We can pass, in a non-deterministically manner, from one configuration of Π to another by applying to its multisets the evolution rules associated with their corresponding membranes. This is done as follows: given a rule $u \to v$ of a membrane i, the objects in u are removed from M_i; then, for every $(ob, out) \in v$ an object ob is put into the multiset associated with the parent membrane (or the external environment if i is the skin membrane); for every $(ob, here) \in v$ an object ob is added to M_i; finally, for every $(ob, in_j) \in v$ an object ob is added to M_j (if j is not a child membrane of i, the rule cannot be applied). Finally, if $\delta \in v$, then the membrane i is dissolved, that is, it is removed from the membrane structure. Moreover, the *priority relation* among the rules forbids the application of a rule if another one of higher priority is applied.

Given two configurations, C and C', of Π, we say that C' is obtained from C in one *transition step*, and we write $C \Rightarrow C'$, if we can pass from the first to the second one by using the evolutions rules appearing in the membrane structure of C in a parallel and maximal way, and for all the membranes at the same time.

Definition 3. *Let Π be a decision P system with external output. A computation, $\mathcal{C}$, of Π with input $m \in M(\Sigma)$ is a sequence, possibly infinite, of configurations of Π, $C_0 \Rightarrow C_1 \Rightarrow \dots \Rightarrow C_q$, $q \geq 0$, such that*

- *C_0 is the initial configuration of Π for m.*
- *Each C_i is obtained from the previous configuration by one transition step.*

We say that $\mathcal{C}$ is a halting computation of Π, if $q \in \mathbb{N}$ and there is no rule applicable to the objects present in its last configuration. In this case, we say that C_q is the halting configuration of $\mathcal{C}$.

We say that Π is deterministic if for each $m \in M(\Sigma)$ there exists an unique computation with input m.

The philosophy of the P systems with external output is that we cannot know what is happening inside the membrane structure, but we can only collect the information thrown from it to the external environment. In accordance with it, it seems natural that the halting computations of these P systems report to the outside when they have reached their final configurations (accepting or rejecting). Furthermore, the idea behind the decision P systems is to use them as languages decision devices. These considerations lead us to the following notions.

Definition 4. *A deterministic decision P system with external output, Π, is said to be valid when the following is verified:*

- *All of its computations halt.*
- *For each computation of Π only one rule of the form $u \rightarrow v(ob, out)$, where $ob = YES$ or $ob = NO$, must have been applied in the skin membrane of μ_{Π}, and only in the last step of the computation.*

Definition 5. *Let Π be a deterministic valid decision P system with external output. We say that a configuration (μ^E, M) of Π is an accepting (resp., rejecting) configuration if $YES \in M_E$ (resp., $NO \in M_E$).*

We say that $\mathcal{C}$ is an accepting (resp., rejecting) computation of Π if its associated halting configuration is an accepting (resp., rejecting) configuration.

Definition 6. *A deterministic valid decision P system with external output, Π, accepts (respectively, rejects) a multiset $m \in M(\Sigma)$ if the computation of Π with input m is an accepting (resp. rejecting) computation.*

4 Simulating Turing Machines by Decision P systems

In what follows we are going to define what is meant by simulating a Turing machine (as a languages decision device) through decision P systems with external output. This has to be done in such a way that every solution for a decision problem by a Turing machine provides a solution for the same problem by a decision P system. Moreover, the additional costs of the reduction from one solution to another must be basically polynomial over the input size.

We take as a model the concept of complexity classes in membranes introduced by G. Păun in [1].

Definition 7. *We say that a deterministic Turing machine, TM, is simulated in polynomial time by a family of deterministic decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \dots, \Pi_{TM}(k), \dots)$ if*

1. *For each $k \in \mathbf{N}$, $\Pi_{TM}(k)$ is a deterministic valid decision P system with external output.*
2. $\mathbf{\Pi_{TM}}$ *is an uniform family; that is, there exists a deterministic Turing machine which constructs $\Pi_{TM}(k)$ in polynomial time starting from $k \geq 1$. So, $\Pi_{TM}(k)$ has a polynomial size in the following sense: the size of the working alphabet, the number of membranes, the size of the initial multisets, and the sum of the lengths of all the rules, is bounded by k^r, for some constant r depending on TM.*
3. *Each $\Pi_{TM}(k)$ is polynomially confluent; that is, there exists a polynomial $p(k)$, depending on TM, such that every computation of $\Pi_{TM}(k)$ always halts in less than $p(k)$ steps.*
4. *Each $\Pi_{TM}(k)$ is sound; that is, TM accepts (resp. rejects) the input string $a_{i_1} \dots a_{i_k}$ if and only if $\Pi_{TM}(k)$ accepts (resp. rejects) $g(a_{i_1} \dots a_{i_k})$ (being g a suitable polynomial encoding of strings by multisets).*

Theorem 1. *Each deterministic Turing machine can be simulated in polynomial time by a family of deterministic decision P systems.*

Proof. We consider deterministic Turing machines following [5].

Suppose we have $Q_{TM} = \{q_N, q_Y, q_0, \dots, q_n\}$, $\Gamma_{TM} = \{B, \triangleright, a_1, \dots, a_m\}$, $\Sigma_{TM} = \{a_1, \dots, a_p\}$, with $p \leq m$, and $\delta_{TM}(q_i, a_j) = (q_{Q(i,j)}, a_{A(i,j)}, D(i,j))$ as set of states, working alphabet, input alphabet and transition function for TM, respectively. We denote $a_B = B$ and $a_0 = \triangleright$.

We construct the family of deterministic decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \dots, \Pi_{TM}(k), \dots)$ which simulates TM as follows: for each $k \in \mathbf{N}$, the decision P system $\Pi_{TM}(k)$ is:

$$\begin{aligned}
\Sigma_k &= \{\langle a, i\rangle : a \in \Sigma_{TM}, 1 \leq i \leq k\} \\
\Gamma_k &= \{\langle a, i\rangle : a \in \Sigma_{TM}, 0 \leq i \leq k\} \cup \{s_i^-, s_i^+, s_i : 1 \leq i \leq 9\} \cup \\
&\quad \{s, q_N, q_Y, h, h', YES, NO\} \cup \{q_i : 0 \leq i \leq n\} \cup \{b_i, b_i', b_i'', c_i : 0 \leq i \leq m\} \\
\mu_\Pi &= [_1\ ]_1, \quad i_\Pi = 1, \quad \mathcal{M}_1 = \{\{q_0, b_0, s_1^-, \dots, s_9^-, s\}\}
\end{aligned}$$

$$R = R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4, \text{where}$$

$$\bullet R_0 \equiv \left.\begin{array}{l} \langle a_i, j\rangle \to \langle a_i, j-1\rangle^2 \quad {\scriptstyle (1 \leq i \leq p, 1 \leq j \leq k)} \\ \langle a_i, 0\rangle \to b_i \quad {\scriptstyle (1 \leq i \leq p)} \end{array}\right\} > s \to s_1$$

$\bullet R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$, with

$$R_{1,1} \equiv s_1 s_1^- \to s_1^+ > s_1^- \to s_1^- > \left\{\begin{array}{l} h \to hh' \\ b_i \to b_i b_i' \quad {\scriptstyle (0 \leq i \leq m)} \\ s_1^+ \to s_1^- s_2 \end{array}\right.$$

$$R_{1,2} \equiv \left\{\begin{array}{l} h' s_2 s_2^- \to s_2^+ > s_2 \to s_3 > s_2^- \to s_2^- > \\ \qquad > {b'}_i^2 \to b_i'' \quad {\scriptstyle (0 \leq i \leq m)} > \left\{\begin{array}{l} b_i' \to \lambda \quad {\scriptstyle (0 \leq i \leq m)} \\ b_i'' \to b_i' \quad {\scriptstyle (0 \leq i \leq m)} \\ s_2^+ \to s_2^- s_2 \end{array}\right. \end{array}\right.$$

$$R_{1,3} \equiv s_3 s_3^- \to s_3^+ > s_3^- \to s_3^- > \begin{cases} b_i'^2 \to \lambda \quad {\scriptstyle (0 \le i \le m)} \\ s_3^+ \to s_3^- s_4 \end{cases}$$

•$R_2 = R_{2,1} \cup R_{2,2}$, with

$$R_{2,1} \equiv s_4 s_4^- \to s_4^+ > s_4^- \to s_4^- > \begin{cases} h \to hh' \\ s_4^+ \to s_4^- s_5 \end{cases}$$

$$R_{2,2} \equiv s_5 s_5^- \to s_5^+ > s_5^- \to s_5^- > \begin{cases} \text{Rules for the transition} \\ \text{function} \\ s_5^+ \to s_5^- s_6 \end{cases}$$

The rules for the transition function, δ_{TM}, are the following:

Case 1: state q_r, element $a_s \neq B$

Movement	*Rules*
left	$q_r b_s' h \to q_{Q(r,s)} b_s' c_{A(r,s)}$, if $A(r,s) \neq B$ $q_r b_s' h \to q_{Q(r,s)} b_s'$, if $A(r,s) = B$
stand	$q_r b_s' \to q_{Q(r,s)} b_s' c_{A(r,s)}$, if $A(r,s) \neq B$ $q_r b_s' \to q_{Q(r,s)} b_s'$, if $A(r,s) = B$
right	$q_r b_s' \to q_{Q(r,s)} b_s' c_{A(r,s)} h$, if $A(r,s) \neq B$ $q_r b_s' \to q_{Q(r,s)} b_s' h$, if $A(r,s) = B$

Case 2: state q_r, no element

Movement	*Rules*
left	$q_r h \to q_{Q(r,s)} c_{A(r,s)}$, if $A(r,s) \neq B$ $q_r h \to q_{Q(r,s)}$, if $A(r,s) = B$
stand	$q_r \to q_{Q(r,s)} c_{A(r,s)}$, if $A(r,s) \neq B$ $q_r \to q_{Q(r,s)}$, if $A(r,s) = B$
right	$q_r \to q_{Q(r,s)} c_{A(r,s)} h$, if $A(r,s) \neq B$ $q_r \to q_{Q(r,s)} h$, if $A(r,s) = B$

To avoid conflicts, every rule in case 1 has higher priority than any rule in case 2.

•$R_3 = R_{3,1} \cup R_{3,2}$, with

$$R_{3,1} \equiv h' s_6 s_6^- \to s_6^+ > s_6 \to s_7 > s_6^- \to s_6^- > \begin{cases} b_i' \to b_i'^2 \quad {\scriptstyle (0 \le i \le m)} \\ c_i \to c_i^2 \quad {\scriptstyle (0 \le i \le m)} \\ s_6^+ \to s_6^- s_6 \end{cases}$$

$$R_{3,2} \equiv s_7 s_7^- \to s_7^+ > s_7^- \to s_7^- > \begin{cases} b_i b_i' \to \lambda \quad {\scriptstyle (0 \le i \le m)} \\ c_i \to b_i \quad {\scriptstyle (0 \le i \le m)} \\ s_7^+ \to s_7^- s_8 \end{cases}$$

•$R_4 = R_{4,1} \cup R_{4,2}$, with

$$R_{4,1} \equiv s_8 s_8^- \to s_8^+ > s_8^- \to s_8^- > \begin{cases} q_Y \to q_Y s_9, \quad q_N \to q_N s_9 \\ q_i \to q_i s_1 \quad {\scriptstyle (0 \le i \le n)} \\ s_8^+ \to s_8^- \end{cases}$$

$$R_{4,2} \equiv \quad s_9 s_9^- \to \lambda > s_9^- \to s_9^- > \begin{cases} s_1^- \dots s_8^- \to \lambda \\ q_Y \to (YES, out), \quad q_N \to (NO, out) \\ h \to \lambda \\ b_i \to \lambda \quad {\scriptstyle (0 \leq i \leq m)} \end{cases}$$

Let us see that $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \dots, \Pi_{TM}(k), \dots)$ is a family of deterministic decision P systems which simulates TM.

Obviously it is a family of deterministic valid P systems. Moreover, $\mathbf{\Pi_{TM}}$ is an *uniform family*. Indeed, let TM be a deterministic Turing machine such that the set of states has size $n + 2$, the working alphabet has size $m + 2$ and the input alphabet has size p (with $p \leq m$). The necessary resources to construct $\Pi_{TM}(k)$ are the following:

1. The size of the working alphabet Γ_k is $p \cdot k + 4m + n + 33 \in \theta(k \cdot (m + n))$.
2. The degree of the P system is 1.
3. The size of the initial configuration for each $x \in \Sigma_{TM}^k$ is $k + 12 \in \theta(k)$.
4. The total number of rules is in the order of $O(p \cdot k + n \cdot m) = O(k \cdot (m + n))$.
5. The greatest length of a rule is $8 \in O(1)$.

Let us see now that, for each $k \in \mathbb{N}$, $\Pi_{TM}(k)$ is *sound*: let L be the language decided by TM. In order to decide if a string $a_{i_1} \dots a_{i_k} \in \Sigma_{TM}$, of size k, belongs to L, we encode it by the multiset $\{\{(a_{i_1}, 1), \dots, (a_{i_k}, k)\}\}$ which is the input given to $\Pi_{TM}(k)$. The rules of this P system have been carefully chosen in such a way that its operation goes through the following main stages:

1. Transform the input multiset into another multiset which encode the string in base 2.
2. Read the element in the cell scanned by the head.
3. Calculate the new element to write in the cell, move the head and change state.
4. Erase the old element and write the new element.
5. Check if a final state is reached: if not, repeat from stage 2; if q_Y is the final state reached, then accept the string; if q_N is the final state reached, then reject the string.

These stages will be carried out in several small steps, each of them managed by a group of rules. To avoid rules from distinct steps being applied together, we will use s_j^- as *forbidding objects* and s_j^+ as *permitting objects*; if s_j^- is present in the P System, then rules for step j *cannot* be executed; if, instead, s_j^+ is the object present, then rules for step j *must* be executed (if possible). Of course, at any time, for each step only the corresponding forbidding or permitting object will be present. Also there will always be only one permitting object at the same time.

To indicate that we want to perform a step, we will use s_j as *promoter objects*. When s_j appears in the P System, it will transform its corresponding forbidding object s_j^- into the permitting one s_j^+, thus allowing the rules for step

j to be applied. Then the permitting object will transform itself again into the forbidding one, and into a suitable promoter object.

In order to simulate the running of the Turing Machine, it is obvious that we have to fix a representation for several elements of it: which is the current state of the Turing Machine, which are the elements in the cells of the tape, and which cell is scanning the head.

To represent the states we will use objects $q_N, q_Y, q_0, \ldots, q_n$.

Through the simulation, objects $b_0, \ldots, b_m$ will represent, in base 2, which cells contain symbols $a_0, \ldots, a_m$, respectively (note that, at any time, the number of non-empty cells in the tapes of the Turing machine is finite); objects $b'_0, \ldots, b'_m, b''_0, \ldots, b''_m$ will be used as working copies of the previous objects; the single prime objects will also serve to save the symbols read from the cells, and objects $c_0, \ldots, c_m$ will serve to indicate the new symbols to write into them.

The cell scanned by the head, numbered from zero, will be represented by the object h, in base one; object h' will be used, when needed, as a working copy of object h; it will be utilized as a counter.

Finally, let us see that, for each $k \in \mathbb{N}$, $\Pi_{TM}(k)$ is *polynomially confluent*. Indeed, if $x \in \Sigma^k_{TM}$ is an input string of size k to the Turing machine then we have:

1. The generation of the multiset coding, in base 2, the non-empty cells in the initial configuration of TM for x requires $k+2$ steps of the P system.
2. The simulation of each transition step of the Turing machine requires a cost in the order of $O(5j+12)$, where j is the cell being read by the head of TM.
3. Checking if a final state has been reached requires 2 steps.
4. The output after the detection of a final state requires 2 steps.

Therefore, if TM accepts or rejects x in $O(t(k))$ steps then $\Pi_{TM}(k)$ needs $O(k+t(k))$ steps to do the same. $\square$

5 Main result

In [6], C. Zandron, C. Ferretti y G. Mauri prove the following result.

Theorem 2. *If $P \neq NP$ then no NP–complete problem can be solved in polynomial time, with regard to the input length, by a deterministic P system with active membranes but without membrane division.*

In this section we prove a kind of reciprocal result of the previous theorem through the solvability of a decision problem by a family of deterministic decision P systems with external output.

Recall that a decision problem, X, is a pair (E_X, f_X) such that E_X is a language over a certain alphabet and f_X is a boolean mapping over E_X. The elements of E_X are called instances of the problem X. For each $k \in \mathbb{N}$ we note E^k_X the language of all the instances of X with size k.

Definition 8. *We say that a decision problem, X, is solvable in polynomial time by a family of deterministic decision P systems $\mathbf{\Pi_X} = (\Pi_X(1), \Pi_X(2), \ldots, \Pi_X(k), \ldots)$ if*

1. *For each $k \in \mathbf{N}$, $\Pi_X(k)$ is a deterministic valid decision P system with external output.*
2. *$\mathbf{\Pi_X}$ is an uniform family; that is, there exists a Turing machine which constructs $\Pi_X(k)$ in polynomial time starting from $k \geq 1$. So, $\Pi_X(k)$ has a polynomial size in the following sense: the size of the working alphabet, the number of membranes, the size of the initial multisets, and the sum of the lengths of all the rules, is bounded by k^r, for some constant r depending on X.*
3. *Each $\Pi_X(k)$ is polynomially confluent; that is, there exists a polynomial $p(k)$ depending on X such that every computation of $\Pi_X(k)$ always halts in less than $p(k)$ steps.*
4. *Each $\Pi_X(k)$ is sound; that is, for every $a \in E_X^k$, $f_X(a) = 1$ if and only if $\Pi_X(k)$ accepts (resp. rejects) the multiset $g(a)$ (being g a suitable polynomial encoding of elements of E_X by multisets).*

Theorem 3. *If there exists an NP-complete problem that cannot be solved in polynomial time, with regard to the input length, by a family of deterministic decision P system with external output, then $P \neq NP$.*

Proof. Let us suppose that $P = NP$. Then, there exists an NP–complete problem, X, and a deterministic Turing machine, TM_X, solving X in polynomial time with regard to the input length. From theorem 1, TM_X is simulated in polynomial time by a family of deterministic decision P systems, $\mathbf{\Pi_{TM_X}}$. Then, according with definition 2, $\mathbf{\Pi_{TM_X}}$ solves the problem X in polynomial time with regard to the input length. What leads to a contradiction.

□

6 Final Remarks

In this work we make clear an apparent theoretical interest of P systems without membrane division as a tool which allows us to attack the $P \neq NP$ conjecture. The search of an *adequate* NP–complete problem and the study of its solvability through such P systems will give us a direct answer to the conjecture. If the considered problem is solvable in polynomial time then the conjecture will have an affirmative answer; on the other case, it will have a negative answer.

We think that this result provides an additional attractiveness to the research of P systems because it allows us to attack the $P \neq NP$ conjecture within this model.

Of course, we cannot get round the real hardness that we found when trying to prove the unsolvability of a problem in an unconventional model of computation, as P systems are.

References

1. Păun, G.: Membrane Computing. An introduction, *Springer-Verlag*, to appear.
2. Păun, G.: Computing with membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108–143.
3. Păun, G., Rozenberg, G.: A guide to membrane computing, *Theoretical Computer Sciences*, to appear.
4. Păun, G., Rozenberg, G., Salomaa, A.: Membrane Computing with External Output, *Fundamenta Informaticae*, **41**(3), 2000, 313–340.
5. Romero-Jiménez, A., Pérez-Jiménez, M.J.: Simulating Turing Machines by P Systems with External Output, *Fundamenta Informaticae*, **49**(1-3), 2002, 273–287.
6. Zandron, C., Ferretti, C., Muri, G.: Solving NP-Complete Problems Using P Systems with Active Membranes, in *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, 2000, 289–301.
7. The P Systems Web Page:
   `http://bioinformatics.bio.disco.unimib.it/psystems/`