Decision P Systems and the P≠NP Conjecture

Mario J. Pérez Jiménez, Álvaro Romero Jiménez,
and Fernando Sancho Caparrini

Abstract. We introduce *decision* P systems, which are a class of P systems with symbol-objects and external output. The main result of the paper is the following: if there exists an NP–complete problem that cannot be solved in polynomial time, with respect to the input length, by a deterministic decision P system 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 respect to the input length, by a deterministic P system with active membranes but without membrane division, constructed in polynomial time from the input. Together, these results give a characterization of $P \neq NP$ in terms of deterministic P systems.

1 Introduction

In [2] a new model of computation, called *P Systems*, is introduced within the framework of *Natural Computing* (bio-inspired computing). 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 starts 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 [8]. In particular, P systems with external output are studied in [4].

The different variants of P systems found in the literature are in general 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 consider here works with symbol–objects and it has two characteristics that have seldom been considered before: we work with *decision* devices whose work is triggered by certain *input data*. The aim is to use this kind of P systems to deal with decision problems.

The main goal of this paper is to show a sufficient condition for the relation $P \neq NP$ to be verified: if there exists an NP–complete problem that cannot be

solved in polynomial time, with respect to the input length, by any family of deterministic decision P systems, *constructed* in polynomial time, then $P \neq NP$.

To achieve this, we prove that every decision problem which can be solved by a deterministic Turing machine in polynomial time can also be solved by a family of 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 (with symbol–objects) of *decision* P systems; Section 4 shows 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 a mapping $m : A \to \mathbb{N}$; $m(a)$ is the number of copies of $a \in A$ in the multiset m. The set $\{a \in A : m(a) > 0\}$ is called the *support* of m and it is denoted by $supp(m)$. A multiset, m, is said to be empty (resp. finite) if its support 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. For two multisets m_1, m_2 over A we define their union by $(m_1 \cup m_2)(a) = m_1(a) + m_2(a)$, for each $a \in A$.

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*. The *degree* of a membrane structure is the number of membranes in it.

The *membrane structure with environment* associated with the membrane structure, μ, is $\mu^E = \big[_E \mu\big]_E$. If we consider μ^E as a rooted tree, then the root node is called the *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*. An evolution rule over Γ is a pair (u, v), usually written $u \to v$, 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 membranes and possibly dissolving the original membrane.

The *length* of a rule is the number of symbols involved in the rule (for instance, the length of $u \to v$ is $|u| + |v| + 1$).

3 Decision P Systems

Definition 1. *A decision P system is a construct*

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

where:

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

To formalize the semantics of this model we define first what a configuration of such a P system is, and then 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 the work of Π.

We can pass, in a non-deterministic 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, then the rule cannot be applied). Finally, if $\delta \in v$, then the membrane i is dissolved (if i is the skin membrane, the rule cannot be applied), that is, it is removed from the membrane structure. The objects of a dissolved membrane remain in the region surrounding it, while the rules are removed. 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 evolution rules appearing in the membrane structure of C in a parallel and maximal way in each membrane, and for all the membranes at the same time.

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

- *C_0 is the initial configuration of Π, with the multiset m placed in membrane i_π.*
- *Each C_i $(1 \leq i \leq q)$ is obtained from the previous configuration by one transition step.*

We say that $\mathcal{C}$ is a halting computation of Π if 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 sent out from it to the environment. Thus, it is natural that the halting computations of these P systems report to the environment 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, Π, is said to be* valid *when the following is verified:*

- *All computations of Π halt.*
- *For each computation of Π only one rule of the form $u \to v(ob, out)$, where $ob = YES$ or $ob = NO$, may be 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. 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, Π, accepts (respectively, rejects) a multiset $m \in M(\Sigma)$ if the computation of Π with input m is an accepting (resp. rejecting) computation.*

We denote by $\mathcal{D}$ the class of all deterministic valid decision P systems.

4 Simulating Turing Machines by Decision P Systems

In what follows we are going to define what we mean by simulating a Turing machine (as a languages generating device) through a family of deterministic decision P systems. This has to be done in such a way that every solution for a decision problem given 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 polynomial in terms of the input size.

We take as a model the concept of complexity classes in membrane systems 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 valid decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ if:*

1. *The family $\mathbf{\Pi_{TM}}$ is $\mathcal{D}$–consistent; that is, for each $k \in \mathbf{N}^+$, $\Pi_{TM}(k)$ is a deterministic valid decision P system.*
2. *The family $\mathbf{\Pi_{TM}}$ is TM–uniform; that is, there exists a deterministic Turing machine, TM', which constructs $\Pi_{TM}(k)$ in polynomial time starting from $k \geq 1$ (there exists a polynomial $p'(k)$ depending on TM such that for each k, $TM'(k)$ halts in less than $p'(k)$ steps and its output is $\Pi_{TM}(k)$).*
3. *The family $\mathbf{\Pi_{TM}}$ is polynomially bounded; 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. *The family $\mathbf{\Pi_{TM}}$ is TM–sound; that is, the Turing machine TM accepts (resp. rejects) the input string $a_{i_1} \ldots a_{i_k}$ if and only if $\Pi_{TM}(k)$ accepts (resp. rejects) $g(a_{i_1} \ldots a_{i_k})$ (g is a suitable polynomial encoding of strings by multisets).*

Note 1. The fact that the family $\mathbf{\Pi_{TM}}$ is $\mathcal{D}$–consistent has the consequence that for each $k \geq 1$, the P system $\Pi_{TM}(k)$ 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.

Note 2. A suitable polynomial encoding, g, of strings by input multisets of $\Pi_{TM}(k)$ means the following: there exists a Turing machine, TM'', and a polynomial $q(k)$ depending on TM such that for each input data w of TM we have that $TM''(w)$ halts in less than $q(|w|)$ steps and its output is $g(w)$ (an input multiset of the P system $\Pi_{TM}(|w|)$).

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

Proof. We consider deterministic Turing machines following [6].

Suppose we have $Q_{TM} = \{q_N, q_Y, q_0, \ldots, q_n\}$, $\Gamma_{TM} = \{B, \triangleright, a_1, \ldots, a_m\}$, $\Sigma_{TM} = \{a_1, \ldots, a_p\}$, 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 a family of deterministic decision P systems $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ which simulates TM as follows: for each $k \in \mathbf{N}$, the decision P system $\Pi_{TM}(k)$ is:

- Input alphabet: $\Sigma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 1 \leq i \leq k\}$
- Working alphabet: $\Gamma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 0 \leq i \leq k\} \cup \{t_i : 1 \leq i \leq k\} \cup \{s_i^-, s_i^+, s_i : i \in \{T_1, T_2, F, S, 1, \ldots, 9\}\} \cup \{q_N, q_Y, h, h', YES, NO\} \cup \{q_i : 0 \leq i \leq n\} \cup \{b_i, b_i', b_i'', c_i : 0 \leq i \leq m\}$
- Membranes structure: $\mu_{\Pi} = [_1\,]_1$.
- Input membrane: $i_{\Pi} = 1$.
- Initial multisets: $\mathcal{M}_1 = \{\{q_0, b_0, s_{T_1}^-, s_{T_2}^-, s_F^-, s_S^-, s_1^-, \ldots, s_9^-, s_{T_1}\}\}$.
- Evolution rules: $R = R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4$, where:

- $R_0 = R_{0,1} \cup R_{0,2} \cup R_{0,3} \cup R_{0,4}$, with

$$R_{0,1} \equiv s_{T_1} s_{T_1}^- \to s_{T_1}^+ > s_{T_1}^- \to s_{T_1}^- > \begin{cases} \langle a_i, j\rangle \to \langle a_i, j\rangle t_j & (1 \leq i \leq p, 1 \leq j \leq k) \\ s_{T_1}^+ \to s_{T_1}^- s_{T_2} \end{cases}$$

$$R_{0,2} \equiv \begin{cases} s_{T_2} s_{T_2}^- \to s_{T_2}^+ > s_{T_2}^- \to s_{T_2}^- > t_1^2 s_{T_2}^+ \to s_{T_2}^- s_F > \cdots > \\ > t_k^2 s_{T_2}^+ \to s_{T_2}^- s_F > t_1 \ldots t_k s_{T_2}^+ \to s_{T_2}^- s_S > s_{T_2}^+ \to s_{T_2}^- s_F \end{cases}$$

$$R_{0,3} \equiv s_F s_F^- \to s_F^+ > s_F^- \to s_F^- > \begin{cases} s_{T_1}^- s_{T_2}^- s_S^- s_1^- \ldots s_9^- s_F^+ q_0 b_0 \to (NO, out) \\ \langle a_i, j\rangle \to \lambda \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ t_j \to \lambda \quad (1 \leq j \leq k) \end{cases}$$

$$R_{0,4} \equiv \begin{cases} s_S s_S^- \to s_S^+ > s_S^- \to s_S^- > \begin{cases} \langle a_i, j\rangle \to \langle a_i, j-1\rangle^2 & (1 \leq i \leq p, 1 \leq j \leq k) \\ \langle a_i, 0\rangle \to b_i & (1 \leq i \leq p) \end{cases} \\ > s_S^+ \to s_S^- s_1 \end{cases}$$

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

$$R_{1,1} \equiv s_1 s_1^- \to s_1^+ > s_1^- \to s_1^- > \begin{cases} h \to h h' \\ b_i \to b_i b_i' \quad (0 \leq i \leq m) \\ s_1^+ \to s_1^- s_2 \end{cases}$$

$$R_{1,2} \equiv \begin{cases} h' s_2 s_2^- \to s_2^+ > s_2 \to s_3 > s_2^- \to s_2^- > \\ \qquad > b_i'^2 \to b_i'' \quad (0 \leq i \leq m) > \begin{cases} b_i' \to \lambda \quad (0 \leq i \leq m) \\ b_i'' \to b_i' \quad (0 \leq i \leq m) \\ s_2^+ \to s_2^- s_2 \end{cases} \end{cases}$$

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

- $R_2 = R_{2,1} \cup R_{2,2}$, 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

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

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

- $R_4 = R_{4,1} \cup R_{4,2}$, with

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

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

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

Obviously it is a family of deterministic valid decision 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 $p \cdot k + 4m + n + 45$; that is, in the order $\theta(k \cdot m + n)$.
2. The degree of the P system is 1.
3. The size of the initial configuration for each $x \in \Sigma_{TM}^k$ is $k + 16 \in \theta(k)$.
4. The total number of rules is in the order of
$$O(p \cdot k + n \cdot m) = O(k \cdot m + n \cdot m)$$
5. The greatest length of a rule is $16 \in O(1)$.

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 length k, belongs to L, we encode it by the multiset $\{\{\langle a_{i_1}, 1\rangle, \dots, \langle a_{i_k}, k\rangle\}\}$ 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 work goes through the following main stages:

1. Check that the multiset received as input codes a string of length k. For this, we have to verify that for each $j = 1, \dots, k$ there exists one and only one pair whose second component is equal to j. Otherwise, the P system halts and rejects the multiset.
2. Transform the input multiset into another multiset which encodes the string in base 2 (in order to specify the symbol written in each cell of the tape).
3. Read the element in the cell scanned by the head.
4. Compute the new element to be writen in the cell, move the head and change state (according to the transition function).
5. Erase the old element and write the new element.
6. 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 exist only one permitting object in the system.

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.

The first two stages take care of filtering the input multisets: those which do not encode a string of size k are rejected; those which do encode such strings are transformed in such a way that we have a code in base 2 of the symbols in the cells of the tape of the Turing machine. These operations are perfomed by the rules in R_0.

For a multiset to correctly encode a string of size k, it has to verify two conditions: for each $j = 1, \ldots, k$, it has to contain no more than one pair with second component equal to j; for each $j = 1, \ldots, k$, it has to contain at least one pair with second component equal to j. These two conditions are checked by rules in $R_{0,1}$ and $R_{0,2}$, using objects t_j as signals for the second components of the pairs.

If the check fails, all objects in the P system are eliminated and it sends out the object NO, to reject the multiset. If the check passes, then we double j times each pair of the form $\langle a, j \rangle$, changing them at the end by a b_j.

For a detailed description of the simulation of the running of the Turing machine over a correct multiset see [6]. We only recall here how we represent the Turing machine inside the P system.

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

During the simulation, objects $b_0, \ldots, b_m$ will represent, in base 2, the cells which 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 be useful to indicate the symbols read from the cells, and objects $c_0, \ldots, c_m$ will be useful 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 used as a counter.

Finally, let us see that the family $\mathbf{\Pi_{TM}}$ is *polynomially bounded*. Indeed, if $x \in \Sigma_{TM}^k$ is an input string of size k of the Turing machine, then we have:

1. The check for a multiset to correctly encode a string of size k needs four steps in the affirmative case and six steps in the negative one (in this last case, the P system halts).
2. The generation of the multiset encoding, in base 2, the non-empty cells in the initial configuration of TM for x requires $k + 3$ steps of the P system.
3. 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.

4. Checking if a final state has been reached requires 2 steps.
5. 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 the P system $\Pi_{TM}(k)$ needs $O(k + t^2(k))$ steps to do the same. □

5 The Main Result

In [7], C. Zandron, C. Ferretti, and 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 valid decision P systems.

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_X^k 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 valid decision P systems $\mathbf{\Pi_X} = (\Pi_X(k))_{k \in \mathbf{N}^+}$ if:*

1. *The family $\mathbf{\Pi_X}$ is $\mathcal{D}$–consistent; that is, for each $k \in \mathbf{N}$, $\Pi_X(k)$ is a deterministic valid decision P system.*
2. *The family $\mathbf{\Pi_X}$ is X–uniform; that is, there exists a Turing machine, TM', and a polynomial $p'(k)$ depending on X such that for each $k \in \mathbf{N}^+$, $TM'(k)$ halts in less than $p'(k)$ steps and its output is the P system $\Pi_X(k)$.*
3. *The family $\mathbf{\Pi_X}$ is polynomially bounded; 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. *The family $\mathbf{\Pi_X}$ is X–sound; that is, for every $a \in E_X^k$, $f_X(a) = 1$ if and only if $\Pi_X(k)$ accepts the multiset $g(a)$ (g is a suitable polynomial encoding of elements of E_X by input multisets of $\Pi_X(k)$).*

Note 3. As in Note 1, from the fact that the family $\mathbf{\Pi_X}$ is $\mathcal{D}$–consistent we infer that for each $k \geq 1$, the P system $\Pi_X(k)$ 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.

Note 4. A suitable polynomial encoding, g, of elements of E_X by input multisets of $\Pi_X(k)$ means the following: there exists a Turing machine, TM'', and a polynomial $q(k)$ depending on X such that for each input data $w \in E_X$ we have that $TM''(w)$ halts in less than $q(|w|)$ steps and its output is $g(w)$ (an input multiset of the P system $\Pi_X(|w|)$.

Note 5. Given a Turing machine, TM, as a languages generating device, we consider the decision problem, X_{TM}, associated with TM, as follows: $X_{TM} = (\Sigma^*_{TM}, f_{TM})$, where $f_{TM}(w) = 1$ if and only if TM accepts w. According with this definition we infer that a Turing machine is simulated in polynomial time by a family of deterministic valid decision P systems, $\mathbf{\Pi_{TM}}$, if and only if the associated decision problem, X_{TM} is solvable in polynomial time by the family $\mathbf{\Pi_{TM}}$.

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 systems, 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 (actually, all NP–complete problem verifies this property). From Theorem 1, TM_X is simulated in polynomial time by a family of deterministic valid decision P systems, $\mathbf{\Pi_{TM_X}}$. Then, according with Definition 8, $\mathbf{\Pi_{TM_X}}$ solves the problem X in polynomial time with regard to the input length. This leads to a contradiction. □

6 Final Remarks

In this paper we made 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; otherwise, 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.

References

1. Păun, G.: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
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*, **287**, 2002, 73–100.
4. Păun, G.; Rozenberg, G.; Salomaa, A.: Membrane Computing with External Output, *Fundamenta Informaticae*, **41**(3), 2000, 313–340.
5. Pérez Jiménez, M.J.; Romero-Jiménez, A.; Sancho Caparrini, F.: *Teoría de la Complejidad en modelos de computación celular con membranas*, Editorial Kronos, Sevilla, 2002.
6. Romero-Jiménez, A.; Pérez-Jiménez, M.J.: Simulating Turing Machines by P Systems with External Output, *Fundamenta Informaticae*, **49**(1-3), 2002, 273–287.

7. Zandron, C.; Ferretti, C.; Mauri, 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-Verlag, London, 2000, 289–301.
8. The P Systems Web Page: `http://psystems.disco.unimib.it/`