A Linear Solution for QSAT with Membrane Creation

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Francisco J. Romero-Campero

Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla, Avda. Reina Mercedes s/n 41012, Sevilla, Spain
{magutier, marper, fran}@us.es

Abstract. The usefulness of P systems with membrane creation for solving **NP** problems has been previously proved (see [2, 3]), but, up to now, it was an open problem whether such P systems were able to solve **PSPACE**-complete problems in polynomial time. In this paper we give an answer to this question by presenting a uniform family of P system with membrane creation which solves the QSAT-problem in linear time.

1 Introduction

The power of P systems as a tool for efficiently solving **NP** problems has been widely proved. Many examples have been proposed in the framework of P systems with active membranes (with polarizations) and in the framework of P systems with membrane creation.

The complexity class of **NP** problems deals with the *time* needed to solve a problem, i.e., **NP** is the class of problems which can be *solved* by a non-deterministic one-tape Turing machine program where the number of steps is polynomially bounded (see [1]). The key of solving such problems in polynomial time by means of P systems is the creation of an exponential amount of workspace (membranes) in polynomial time.

When we consider the resources needed in a computation, we obviously have to consider the *time*, i.e., the number of steps of our device, but in practice, we also need to consider the amount of memory or storage required by the computation. If we consider a Turing machine computation, the *space* is the number of distinct tape squares visited by the write-read head of the machine. Since the number of visited squares cannot be greater than the number of steps in the computation, we have that, if the number of steps is polynomially bounded, then the number of visited squares is also polynomially bounded. Therefore, any problem solvable in polynomial time is also solvable in polynomial space.

PSPACE (respectively, **NPSPACE**) is the class of decision problems that are solvable by a deterministic (respectively, non–deterministic) Turing machine using a polynomial amount of space. These complexity classes are closed under

polynomial time reduction. Savitch's theorem says that each non–deterministic Turing machine using $f(n)$ space can be simulated by a deterministic Turing machine using only $f(n)^2$ space (for time complexity, such a simulation seems to require an exponential increase in time). Bearing in mind that a Turing machine running in $f(n) \geq n$ time can use at most $f(n)$ space we have $\mathbf{P} \subseteq \mathbf{PSPACE}$ and $\mathbf{NP} \subseteq \mathbf{NPSPACE}$. So, $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE}$. It is unknown whether any of these containments are strict.

A decision problem in **PSPACE** such that every problem in **PSPACE** is polynomial time reducible to it, is called **PSPACE**–complete. If a **PSPACE**–complete problem belongs to **P** (respectively, **NP**), then $\mathbf{P} = \mathbf{PSPACE}$ (respectively, $\mathbf{NP} = \mathbf{PSPACE}$).

In this paper, we present the first polynomial time solution to the QSAT problem, a well known **PSPACE**-complete problem (see L.J. Stockmeyer and A.R. Meyer in [14]) using a family of recognizer P systems with membrane creation. Taking into account that the class of all decision problems solvable in polynomial time by a family of such P systems is closed under polynomial–time reduction, this result shows that all **PSPACE** problems can be solved in polynomial time by P systems with membrane creation.

The paper is organized as follows. In the next section, recognizer P systems are briefly described. In Section 3 the variant of P systems with membrane creation are recalled with a short discussion about their semantics. A linear–time solution to the QSAT problem is presented in the following section, with a short overview of the computation. Finally, some conclusions are given in the last section.

2 Recognizer P Systems

Recognizer P systems were introduced in [13] and are the natural framework to study and solve decision problems, since deciding whether an instance of a problem has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizer P systems are associated with P systems with *input* in a natural way. The data related to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment. In this way, P systems with input and external output are devices which can be seen as black boxes, in which the user provides the data before the computation starts and the P system sends to the environment the output in the last step of the computation. Another important feature of P systems is the non-determinism. The design of a family of recognizer P system has to consider it, because all possibilities in the non-deterministic computations must produce the same answer. This can be summarized in the following definitions.

Definition 1. *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \ldots, p$*, and initial multisets* $w_1, \ldots, w_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ*; the initial multisets are over* $\Gamma - \Sigma$*; and (c)* i_Π *is the label of a distinguished (input) membrane.*

Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, w_1, \ldots, w_{i_\Pi} \cup m, \ldots, w_p)$.

Definition 2. *A* recognizer P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output such that:*

1. *The working alphabet contains two distinguished elements* `yes`, `no`.
2. *All computations halt.*
3. *If* $\mathcal{C}$ *is a computation of* Π*, then either the object* `yes` *or the object* `no` *(but not both) must have been released into the environment, and only in the last step of the computation.*

We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object `yes` (respectively, `no`) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.

Definition 3. *Let* $\mathcal{F}$ *be a class of recognizer P systems. We say that a decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbb{N}}$*, of* $\mathcal{F}$*, and we denote this by* $X \in \mathbf{PMC}_{\mathcal{F}}$*, if the following holds:*

- *The family* $\mathbf{\Pi}$ *is* polynomially uniform by Turing machines*, that is, there exists a deterministic Turing machine constructing* $\Pi(n)$ *from* $n \in \mathbb{N}$ *in polynomial time.*
- *There exists a pair* (cod, s) *of polynomial-time computable functions over* I_X *such that:*
  - *for each instance* $u \in I_X$*,* $s(u)$ *is a natural number and* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$*;*
  - *the family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, cod, s)*, that is, there exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and, moreover, it performs at most* $p(|u|)$ *steps;*
  - *the family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$*;*
  - *the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of a system with the *same* input must always give the *same* answer.

It can be proved that $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement (see [13]). In this paper we will deal with the class $\mathcal{MC}$ of recognizer P systems with membrane creation.

3 P Systems with Membrane Creation

In this section we recall the description of cellular devices (P systems) with membrane creation.

Basically, a P system[1] consists of a hierarchical membrane structure where each membrane has associated a multiset of objects and a set of rules expressing how these objects can evolve. The *membrane structure* of a P system is a hierarchical arrangement of membranes embedded in a *skin* membrane, which separates the system from its *environment*. A membrane without any membrane inside is called *elementary*. Each membrane determines a *region* (the space enclosed between the membrane and the membranes immediately inside it), which can contain a multiset of *objects*. Associated with the regions there are *rules* that can transform and move those objects.

There are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation, see [5]). Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the new workspace is generated by membrane division, and *P systems with membrane creation*, where the new membranes are created from objects. Both models have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. P systems with active membranes have been successfully used to design solutions to **NP**-complete problems, as SAT [13], Subset Sum [10], Knapsack [11], Bin Packing [12], and Partition [4], but as Gh. Păun pointed out in [9] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. The first results in this way have recently appeared, showing that **NP** problems can also be solved in this framework (see [2, 3]).

Recall that a *P system with membrane creation* is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$, where:

1. $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects* and H is a finite set of *labels* for membranes;
2. μ is a *membrane structure* consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H and $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
3. R is a finite set of *rules*, of the following forms:
   (a) $[a \to v]_h$ where $h \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.
   (b) $a[\ ]_h \to [b]_h$ where $h \in H$, $a, b \in O$. These are *send-in communication rules*. An object is introduced in the membrane, possibly modified.
   (c) $[a]_h \to [\ ]_h\, b$ where $h \in H$, $a, b \in O$. These are *send-out communication rules*. An object is sent out of the membrane, possibly modified.

[1] A layman-oriented introduction can be found in [8], a comprehensive monograph in [7], and the latest information about P systems is available at [15].

(d) $[a]_h \to b$ where $h \in H$, $a, b \in O$. These are *dissolution rules*. In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.
(e) $[a \to [v]_{h_2}]_{h_1}$ where $h_1, h_2 \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *creation rules*. In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

Rules are applied according to the following principles:

- Rules from (a) to (d) are used as usual in the framework of membrane computing, that is, in a maximally parallel way. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions indicated below).
- Rules of type (e) are used also in a maximally parallel way. Each object a in a membrane labelled with h_1 produces a new membrane with label h_2 placing in it the multiset of objects described by the string v.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- The rules associated with the label h are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.
- Several rules can be applied to different objects in the same membrane simultaneously. The exception are the rules of type (d) since a membrane can be dissolved only once.

We denote by $\mathcal{MC}$ the class of recognizer P systems with membrane creation.

4 Solving QSAT in Linear Time

In this section we design a family of recognizer P systems with membrane creation (and using dissolution rules) which solves the QSAT problem (the quantified satisfiability problem).

Given a Boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, with Boolean variables $x_1, \ldots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ (where Q_n is $\exists$ if n is odd, and Q_n is $\forall$, otherwise) is said to be the (existential) *fully quantified* formula associated with $\varphi(x_1, \ldots, x_n)$.

We say that φ^* is satisfiable if there exists a truth assignment, σ, over $\{i \mid 1 \leq i \leq n \wedge i \text{ odd}\}$ such that each extension, σ^*, of σ over $\{1, \ldots, n\}$ verify $\sigma^*(\varphi(x_1, \ldots, x_n)) = 1$.

The QSAT problem is the following one: *Given a Boolean formula* $\varphi(x_1, \ldots, x_n)$ *in conjunctive normal form, determine whether or not the (existential) fully quantified formula* $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ *is satisfiable.*

It is well known that QSAT is a **PSPACE**–complete problem [6].

Next, we provide a polynomial time solution of QSAT by a family of recognizer P systems with membrane creation and *using dissolution rules*, according to Definition 3. We will address the resolution via a brute force algorithm, in the framework of recognizer P systems with membrane creation, which consists in the following phases:

- Generation and Evaluation Stage: Using membrane creation we will generate all possible truth assignments associated with the formula and evaluate it on each one. Specifically, we construct a binary complete tree where the leaves encode all possible truth assignment associated with the formula, and the nodes whose level is even (respectively, odd) are codified by an OR gate (respectively, AND gate). In this stage, the values of the formula corresponding to each truth assignment is obtained in the leaves.
- Checking Stage: In each membrane we check whether or not the formula evaluates true on the truth assignment associated with it. Specifically, we proceed to compute the output of that Boolean circuit (that only have gates AND, OR) from the inputs obtained in the leaves by propagating values along the wires and computing the respective gates until the output gate (the root of the tree) has assigned a value.
- Output Stage: The system sends out to the environment the right answer according to the result of the previous stage.

Let us consider the pair function $\langle\ ,\ \rangle$ defined by $\langle n, m\rangle = ((n+m)(n+m+1)/2)+n$. This function is polynomial-time computable (it is primitive recursive and bijective from $\mathbb{N}^2$ onto $\mathbb{N}$). For any given Boolean formula, $\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$, in conjunctive normal form, with n variables and m clauses, we construct a P system $\Pi(\langle n, m\rangle)$ processing the (existential) fully quantified formula φ^* associated with φ (when an appropriate input is supplied). The family presented here is:

$$\mathbf{\Pi} = \{(\Pi(\langle n, m\rangle), \Sigma(\langle n, m\rangle), i(\langle n, m\rangle)) \mid (n, m) \in \mathbb{N}^2\}.$$

For each element of the family, the input alphabet is

$$\Sigma(\langle n, m\rangle) = \{x_{i,j}, \overline{x}_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$$

the input membrane is $i(\langle n, m\rangle) = t$, and the P system

$$\Pi(\langle n, m\rangle) = (\Gamma(\langle n, m\rangle), H(\langle n, m\rangle), \mu, w_s, w_{<t,\vee>}, R(\langle n, m\rangle))$$

is defined as follows:

- Working alphabet:

$$\begin{aligned}\Gamma(\langle n,m\rangle) = \Sigma(\langle n,m\rangle)\\ \cup\ \{z_{j,c} \mid j \in \{0,\dots n\},\ c \in \{\wedge,\vee\}\}\\ \cup\ \{z_{j,c,l} \mid j \in \{0,\dots,n-1\},\ c \in \{\wedge,\vee\}\ l \in \{t,f\}\}\\ \cup\ \{x_{i,j,l}, \overline{x}_{i,j,l} \mid j \in \{1,\dots,n\},\ i \in \{1,\dots,m\},\ l \in \{t,f\}\}\\ \cup\ \{x_{i,j} \mid j \in \{1,\dots,n\},\ i \in \{1,\dots,m\}\}\\ \cup\ \{r_i, r_{i,t}, r_{i,f} \mid i \in \{1,\dots,m\}\}\\ \cup\ \{d_1,\dots,d_m, q, t_0,\dots,t_4, ans_0,\dots,ans_5, \mathtt{yes}, \mathtt{no}\}\\ \cup\ \{yes_\vee, yes^*, no_\vee, \underline{no}_\vee, yes_\wedge, no_\wedge, no^*, yes_\wedge, \underline{yes}_\wedge, no_\vee, \underline{no}_\wedge\}\\ \cup\ \{YES, NO\}.\end{aligned}$$

- The set of labels, $H(\langle n,m\rangle)$, is

$$\{< l,c >\mid l \in \{t,f\}, c \in \{\wedge,\vee\}\} \cup \{a,s,1,\dots,m\}.$$

- Initial membrane structure: $\mu = [\ [\ ]_{<t,\vee>}\ ]_s$.
- Initial multiset: $w_s = \emptyset$, $w_{<t,\vee>} = \{z_{0,\wedge,t}\ z_{0,\wedge,f}\}$.
- Input membrane: $i(\langle n,m\rangle =< t,\vee >$.
- The set of evolution rules, $R(\langle n,m\rangle)$, consists of the following rules (recall that λ denotes the empty string and if c is $\wedge$ then $\overline{c}$ is $\vee$ and if c is $\vee$ then $\overline{c}$ is $\wedge$):

1. $$\left.\begin{array}{l}[z_{j,c} \to z_{j,c,t}, z_{j,c,f}]_{<l,\overline{c}>}\\ [z_{j,c,l} \to [z_{j+1,\overline{c}}]_{<l,c>}]_{<l',\overline{c}>}\end{array}\right\} \text{for } \begin{array}{l} l,l' \in \{t,f\}, \qquad c \in \{\vee,\wedge\},\\ j \in \{0,\dots,n-1\}.\end{array}$$

The goal of these rules is to create one membrane for each truth assignment to the variables of the formula. Firstly, the object $z_{j,c}$ evolves to two objects, one for the assignment *true* (the object $z_{j,c,t}$), and a second one for the assignment *false* (the object $z_{j,c,f}$). In a second step these objects will create two membranes. The new membrane with t in its label represents the assignment $x_{j+1} = true$; on the other hand, the new membrane with f in its label represents the assignment $x_{j+1} = false$.

2. $$\left.\begin{array}{l}[x_{i,j} \to x_{i,j,t}x_{i,j,f}]_{<l,c>}\\ [\overline{x}_{i,j} \to \overline{x}_{i,j,t}\overline{x}_{i,j,f}]_{<l,c>}\\ [r_i \to r_{i,t}r_{i,f}]_{<l,c>}\end{array}\right\} \text{for } \begin{array}{ll} l \in \{t,f\} & i \in \{1,\dots,m\},\\ c \in \{\vee,\wedge\} & j \in \{1,\dots,n\}.\end{array}$$

These rules duplicate the objects representing the formula so it can be evaluated on the two possible assignments, $x_j = true$ ($x_{i,j,t}, \overline{x}_{i,j,t}$) and $x_j = false$ ($x_{i,j,f}, \overline{x}_{i,j,f}$). The objects r_i are also duplicated ($r_{i,t}, r_{i,f}$) in order to keep track of the clauses that evaluate true on the previous assignments to the variables.

3. $$\left.\begin{array}{l}x_{i,1,t}[\,]_{<t,c>} \to [r_i]_{<t,c>}\\ \overline{x}_{i,1,t}[\,]_{<t,c>} \to [\lambda]_{<t,c>}\\ x_{i,1,f}[\,]_{<f,c>} \to [\lambda]_{<f,c>}\\ \overline{x}_{i,1,f}[\,]_{<f,c>} \to [r_i]_{<f,c>}\end{array}\right\} \text{for } \begin{array}{l} i \in \{1,\dots,m\},\\ c \in \{\vee,\wedge\}.\end{array}$$

According to these rules the formula is evaluated in the two possible truth assignments for the variable that is being analyzed. The objects $x_{i,1,t}$ (resp. $\overline{x}_{i,1,f}$) get into the membrane with t in its label (resp. f) being transformed into the objects r_i representing that the clause number i evaluates true on the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$). On the other hand, the objects $\overline{x}_{i,1,t}$ (resp. $x_{i,1,t}$) get into the membrane with f in its label (resp. t) producing no objects. This represents that these objects do not make the clause true in the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$).

4. $$\left.\begin{array}{l} x_{i,j,l}[\,]_{<l,c>} \to [x_{i,j-1}]_{<l,c>} \\ \overline{x}_{i,j,t}[\,]_{<l,c>} \to [\overline{x}_{i,j-1}]_{<l,c>} \\ r_{i,t}[\,]_{<l,c>} \to [r_i]_{<l,c>} \end{array}\right\} \text{for } \begin{array}{l} l \in \{t,f\}, \quad i \in \{1,\ldots,m\}, \\ c \in \{\vee,\wedge\}, \quad j \in \{2,\ldots,n\}. \end{array}$$

In order to analyze the next variable the second subscript of the objects $x_{i,j,l}$ and $\overline{x}_{i,j,l}$ are decreased when they are sent into the corresponding membrane labelled with l. Moreover, following the last rule, the objects $r_{i,l}$ get into the new membranes to keep track of the clauses that evaluate true on the previous truth assignments.

5. $[z_{n,c} \to d_1 \ldots d_m q]_{<l,\overline{c}>}$ } for $l \in \{t,f\}$ and $c \in \{\vee,\wedge\}$.

At the end of the generation stage the object z_n will produce the objects $d_1,\ldots,d_m$ and yes_0, which will take part in the checking stage.

6. $$\left.\begin{array}{ll} [d_i \to [t_0]_i]_{<l,c>} & \\ r_{i,t}[\,]_i \to [r_i]_i & [r_i]_i \to \lambda \\ [t_s \to t_{s+1}]_i & [t_2]_i \to t_3 \end{array}\right\} \text{for } \begin{array}{l} i \in \{1,\ldots,m\}, \\ s \in \{0,1\}, \quad c \in \{\vee,\wedge\}. \end{array}$$

Following these rules each object d_i creates a new membrane with label i where the object t_0 is placed; this object will act as a counter. The object r_i gets into the membrane labelled with i and dissolves it preventing the counter, t_i, from reaching the object t_2. The fact that the object t_2 appears in a membrane with label i means that there is no object r_i, that is, the clause number i does not evaluate true on the truth assignment associated with the membrane; therefore neither does the formula evaluate true on the associated truth assignment.

7. $$\left.\begin{array}{ll} [q \to [ans_0]_a]_{<l,c>} & \\ t_3[\,]_a \to [t_4]_a & [t_4]_a \to \lambda \\ [ans_h \to ans_{h+1}]_a, & [ans_5]_a \to \texttt{yes} \\ [ans_5 \to \texttt{no}]_{<l,c>} & \end{array}\right\} \text{for } \begin{array}{ll} l \in \{t,f\} & c \in \{\vee,\wedge\}, \\ h = 0,\ldots,4. & \end{array}$$

The object q creates a membrane with label a where the object ans_0 is placed. The object ans_h evolves to the object ans_{h+1}; at the same time the objects t_3 can get into the membrane labelled with a and dissolve it preventing the object `yes` from being sent out from this membrane.

8. $$\left.\begin{array}{ll} [yes]_{<l,c>} \to yes_{\overline{c}} & [no]_{<l,c>} \to no_{\overline{c}} \\ [yes_\vee]_{<l,\vee>} \to yes^* & [no_\vee \to \underline{no}_\vee]_{<l,\vee>} \\ [yes^* \to yes_\wedge]_{<l,\wedge>} & [\underline{no}_\vee]_{<l,\vee>} \to no_\wedge \\ [\underline{no}_\vee \to \lambda]_{<l,\wedge>} & [yes_\vee \to \lambda]_{<l,\wedge>} \\ [no_\wedge]_{<l,\wedge>} \to no^* & [yes_\wedge \to \underline{yes}_\wedge]_{<l,\wedge>} \\ [no^* \to no_\vee]_{<l,\vee>} & [\underline{yes}_\wedge]_{<l,\wedge>} \to yes_\vee \\ [\underline{no}_\wedge \to \lambda]_{<l,\vee>} & [\underline{yes}_\wedge \to \lambda]_{<l,\vee>} \\ [yes^*]_s \to \texttt{yes}\,[\,]_s & [no_\wedge]_s \to \texttt{no}\,[\,]_s \end{array}\right\} \text{for } l \in \{t, f\}.$$

This set of rules controls the output stage. After the evaluation stage, from each working membrane we obtain an object *yes* or *no* depending on whether the truth assignment associated with this membrane satisfies or not the formula. On the contrary to the SAT problem, in QSAT it is not enough that one truth assignment satisfies the formula, but the final answer is YES if an appropriate combination of truth assignments according to the quantifiers $\exists$ and $\forall$ are founded.

4.1 An Overview of the Computation

First of all we define a polynomial encoding of the QSAT problem in the family $\mathbf{\Pi}$ constructed in the previous section. Given a Boolean formula in conjunctive normal form, $\varphi = C_1 \wedge \cdots \wedge C_m$ such that $Var(\varphi) = \{x_1, \ldots, x_n\}$, and being φ^* the (existential) fully quantified formula associated with it, we define $s(\varphi^*) = \langle n, m\rangle$ (recall the bijection mentioned in the previous section) and $cod(\varphi^*) = \{x_{i,j} \mid x_j \in C_i\} \cup \{\overline{x}_{i,j} \mid \neg x_{i,j} \in C_i\}$.

Next we describe informally how the recognizer P system with membrane creation $\Pi(s(\varphi^*))$ with input $cod(\varphi^*)$ works.

In the initial configuration we have the input multiset $cod(\varphi)$ and the objects $z_{0,\wedge,t}$ and $z_{0,\wedge,f}$ placed in the input membrane (membrane labelled with $< t, \vee >$). In the first step of the computation the object $z_{0,\wedge,t}$ creates a new membrane with label $< t, \vee >$ which represents the assignment $x_1 = true$ and the object $z_{0,\wedge,f}$ creates a new membrane with label $< f, \vee >$ which represents the assignment $x_1 = false$. The second component of the labels, i.e., $\wedge$ and $\vee$ will be used in the output stage.

In these two new membranes the object $z_{1,\vee}$ is placed. At the same time the input multiset representing the formula φ is duplicated following the two first rules in group 2. In the next step, according to the rules in group 3, the formula is evaluated on the two possible truth assignments for x_1. In the same step the rules in group 4 decrease the second subscript of the objects representing the formula ($x_{i,j,l}, \overline{x}_{i,j,l}$ with $j \geq 2$) in order to analyze the next variable. Moreover, at the same time, the object $z_{1,c}$ produces the object $z_{1,c,t}$ and $z_{1,c,f}$ ($c \in \{\wedge, \vee\}$) and the system is ready to analyze the next variable. In this way, the generation and evaluation stages go until all the possible assignments to the variables are generated and the formula is evaluated on each one of them. Observe that it takes

two steps to generate the possible assignments for a variable and to evaluate the formula on them; therefore the generation and evaluation stages take $2n$ steps.

The checking stage starts when the object $z_{n,c}$ produces the objects $d_1, \ldots, d_m$ and the object q. In the first step of the checking stage each object d_i, for $i = 1, \ldots, m$, creates a new membrane labelled with i where the object t_0 is placed, and the object q creates a new membrane with label a placing the object yes_0 in it.

The objects $r_{i,t}$, which indicate that the clause number i evaluates true on the truth assignment associated with the membrane, are sent into the membranes by the last rule in group 4 so the system keeps track of the clauses that are true. The objects $r_{i,t}$ get into the membrane with label i and dissolves it in the following two steps preventing the counter t_2 from dissolving the membrane and producing the object t_3 according to the last rule in group 6. If for some i there is no object r_i (this means that the clause i does not evaluate true on the associated assignment) the object t_2 will dissolve the membrane labelled with i producing the object t_3 that will get into the membrane with label a where the object ans_h evolves following the rules in 7. The object t_4 dissolves the membrane with label a preventing the production of the object ans_5. Therefore the checking stage takes 6 steps.

Finally the output stage takes place according to the rules in group 8. If some object ans_5 is present in any membrane with label $< l, c >, (l \in \{t, f\}, c \in \{\wedge, \vee\})$, this means that there exists at least one clauses not satisfied by the truth assignment associated with the membrane, and by the last rule in group 7, we obtain no in this membrane. Otherwise, the object ans_5 will be inside the membrane with label a, it will dissolve the membrane, and send yes to the working membrane.

At this point, in each of the 2^n working membranes we have an object yes or no depending on if the associated truth assignment satisfies or not the formula φ. In the last steps we control the flow of the objects yes and no from the working membranes to the environment. Basically, the process is the following. If there are one object yes inside a membrane with $\vee$ in its label, this object dissolves the membrane and sends out another yes. If this does not happen, i.e., if two objects no are inside a membrane with label $\vee$, the membrane is dissolved and no is sent out. Analogously, if there are one object no inside a membrane with $\wedge$ in its label, this object dissolves the membrane and sends out another no. Otherwise, if two objects yes are inside a membrane with label $\vee$, the membrane is dissolved and yes is sent out.

Consequently, the family $\mathbf{\Pi}$ of recognizer P systems with membrane creation using dissolution rules solves in polynomial time QSAT according to Definition 3. Hence, we have:

Theorem 1. QSAT $\in \mathbf{PMC}_{\mathcal{MC}}$

From this theorem we deduce the following result:

Corollary 1. PSPACE $\subseteq \mathbf{PMC}_{\mathcal{MC}}$

Proof. It suffices to make the following remarks: the QSAT problem is **PSPACE**–complete, QSAT $\in \mathbf{PMC}_{\mathcal{MC}}$, and the complexity class $\mathbf{PMC}_{\mathcal{MC}}$ is closed under polynomial time reduction. □

5 Conclusions and Future Work

P systems are computational devices whose power has to be studied in a deeper extent. In the last time, several paper have explored this power, both in the framework of P systems with active membranes and P systems with membrane creation. These papers have shown that **NP**-complete problems are solvable (in polynomial time) by families of recognizer of P systems of these types, according to Definition 3. In this paper we have shown that **PSPACE**–complete problems can also be solved (in polynomial time) by families of recognizer P systems with membrane creation, in a uniform way.

Both models (active membranes and membrane creation) have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. The specific techniques for designing solutions to concrete problems (generation, evaluation, checking, and output stages) are quite different, so the simulation of one model in the other one is not a trivial question. This seems an interesting open problem to be considered in the future.

Acknowledgement

This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds. F.J. Romero-Campero also acknowledges a FPU fellowship from the same Ministerio.

References

1. M.R. Garey, D.S. Johnson: *Computers and Intractability A Guide to the theory of NP-Completeness.* W.H. Freeman and Company, 1979.
2. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution of Subset Sum Problem by using Mmmbrane creation. In *Mechanisms, Symbols and Models Underlying Using Cognition, First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005* (J. Mira, J.R. Alvarez, eds.). LNCS 3561, Springer, Berlin, 2005, 258–267.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with Membrane Creation. In *Computability in Europe 2005, CiE 2005: New Computational Paradigms* (S. Barry Cooper, B. Lowe, L. Torenvliet, eds.), Report ILLC X-2005-01, University of Amsterdam, 82–91.
4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A fast P system for finding a balanced 2-partition. *Soft Computing*, 9, 9 (2005), 673–678.
5. P.L. Luisi: The chemical implementation of autopoiesis, *Self-Production of Supramolecular Structures* (G.R. Fleishaker et al., eds.), Kluwer, Dordrecht, 1994.
6. C.H. Papadimitriou: *Computational Complexity.* Addison-Wesley, Reading, Mass., 1994.
7. Gh. Păun: *Membrane Computing. An Introduction*, Springer, Berlin, 2002.
8. Gh. Păun, M.J. Pérez-Jiménez: Recent computing models inspired from biology: DNA and membrane computing, *Theoria*, 18, 46 (2003), 72–84.

9. Gh. Păun: Further open problems in membrane computing. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.
10. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by active membranes. *New Generation Computing*, 23, 4 (2005), 367–384.
11. M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear solution for the Knapsack problem using active membranes. *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, July 2003, Revised Papers* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933, Springer, Berlin, 2004, 250–268.
12. M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving the BIN PACKING problem by recognizer P systems with active membranes. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. In *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003* (E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl, eds.), 2003, 284–294.
14. L.J. Stockmeyer, A.R. Meyer: Word problems requiring exponential time. *Proc. 5th ACM Symp. on the Theory of Computing*, 1973, 1–9.
15. P systems web page `http://psystems.disco.unimib.it/`