A Characterization of PSPACE with Antimatter and Membrane Creation

Zsolt Gazdag[1], Miguel A. Gutiérrez–Naranjo[2]

[1]Department of Algorithms and their Applications
Faculty of Informatics
Eötvös Loránd University, Hungary
gazdagzs@inf.elte.hu

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es

Summary. The use of negative information provides a new tool for exploring the limits of P systems as computational devices. In this paper we prove that the combination of antimatter and annihilation rules (based on the annihilation of physical particles and antiparticles) and membrane creation (based on autopoiesis) provides a P system model able to solve **PSPACE**-complete problems. Namely, we provide a uniform family of P system in such P system model which solves the satisfiability problem for quantified Boolean formulas (**QSAT**). In the second part of the paper, we prove that all the decision problems which can be solved with this P system model belong to the complexity class **PSPACE**, so this P system model characterises **PSPACE**.

1 Introduction

The use of *negative information* provides a new challenge in the development of theoretical aspects in Membrane Computing (see [20]). Such *negative information* can be considered by extending the definition of a multiset f on a set X from $f : X \to \mathbb{N}$ to $f : X \to \mathbb{Z}$ (i.e., admitting negative multiplicity of the elements of the multiset [4, 13]) or even considering negative time and the possibility of travelling in time [7].

One of the most extended uses of negative information in Membrane Computing is considering *anti-spikes* in the framework of Spiking Neural P systems. In such model when one *spike* and one *anti-spike* appear in the same neuron, the annihilation occurs and both, spike and anti-spike, disappear [15, 17, 22, 24]. The use of antimatter, as an extension of the concept of anti-spikes, is being explored in other P system models [1, 2, 5].

Recently, it has been proved that antimatter and annihilation rules are a frontier of tractability in Membrane Computing [5]. The starting point for the study was a well-known result in the complexity theory of Membrane Computing: P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary membranes (denoted by $\mathcal{AM}^0_{-d,+ne}$) can solve exactly problems in the complexity class **P** (see [8], Th. 2). The main result in [5] is that $\mathcal{AM}^0_{-d,+ne}$ endowed with antimatter and annihilation rules (denoted by $\mathcal{AM}^0_{-d,+ne,+ant}$) can solve **NP**-complete problems.

In a certain sense, such results show that if the number of membranes of the P system can be increased by membrane division, then endowing the model with dissolution or annihilation rules, then the model is capable to solve **NP**-complete problems.

Similar results hold in the case of P systems with membrane creation. In [9] it is shown that these P systems when dissolution rules are allowed can solve **PSPACE**-complete problems (i.e, they can solve all the decision problems which can be solved by Turing machines, deterministic or non-deterministic, in polynomial space). In this paper, we show that using annihilation rules instead of dissolution rules, P systems with membrane creation are not only able to solve **NP**-complete problems, but **PSPACE**-complete problems too. By taking [23] as starting point, in the second part of the paper, we prove that all the decision problems which can be solved with this P system model belong to the complexity class **PSPACE**, so this P system model characterises **PSPACE**.

The paper is organized as follows. In the next section, the notion of P systems with membrane creation and annihilation rules is introduced. Then recognizer P systems are briefly described. In Section 4 we show that the well known QSAT problem (i.e., the problem of deciding if a fully quantified Boolean formula is true or not) can be solved in linear time by P systems with membrane creation, with annihilation rules and without dissolution rules. In Section 5, we prove that **PSPACE** is an upper bound for the set of decision problems which can be solved with this model. Finally, some conclusions are given in the last section.

2 The P System Model

The basis of the model is two types of rules which are not so common on complexity studies in Membrane Computing. The first type, rules of membrane creation, is based on the biological process of autopoiesis [14]. It creates a membrane from a single object in a similar way to the creation of a vesicle in a cell by a metabolite. This type of rule was first considered in [12, 16] and it has been proved that P systems with membrane creation and *dissolution rules* can solve **NP**-complete problems (see [10, 11]) or even **PSPACE**-complete problems (see [9]).

The idea of using antimatter as a generalization of the anti-spikes used in Spiking Neural P Systems was firstly proposed in [21]. Based on the physical inspiration of particles and antiparticles, if an object a and its opposite one $\overline{a}$

appears simultaneously in the same membrane, they are annihilated by application of the corresponding rule $a\overline{a} \to \lambda$. As pointed above, several authors have started to explore the possibilities of using antimatter in Membrane Computing [1, 2, 5].

Formally, a *P system with membrane creation and annihilation rules* is a construct of the form $\Pi = (O, H, \mu, w_1, \dots, w_m, R)$, 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 $w_1, \dots, w_m$ 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.
   (d) $[a \to [v]_{h_2}]_{h_1}$ 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.
   (e) $[\,a\overline{a} \to \lambda\,]_h$ for $h \in H$, $a, \overline{a} \in O$. This is an annihilation rule, associated with a membrane labelled by h: the pair of objects $a, \overline{a} \in O$ belonging simultaneously to this membrane disappears.

Rules are applied according to the following principles:

- Rules of type (a) - (d) are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non–deterministic way), but any object which can evolve by one rule of any form, must evolve.
- If an object can trigger two or more rules, one of such rules is non-deterministically chosen, except for annihilation rules (type (e)). Any annihilation rule has priority over all rules of the other types of rules. This fact has a clear physical inspiration. If a particle and its antiparticle meet, they *do* disappear and no other option is possible. This semantics was also used in [5].
- 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.

Following the standard notations, the class of these P systems is denoted by $\mathcal{AM}^0_{-d,+mc,+antPri}$, where $-d$ indicates that dissolution rules are not used, $+mc$ indicates the use of membrane creation and we add $+antPri$ to denote the use of antimatter and annihilation rules with priority.

3 Recognizer P Systems

We recall the main notions related to recognizer P systems and complexity in Membrane Computing. For a detailed description see, e.g., [18, 19].

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*) and θ_X is a total Boolean function over I_X. A *P system with input* is a tuple (Π, Σ, i_Π), where Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; and i_Π is the label of a distinguished (input) membrane. Let (Π, Σ, i_Π) be a P system with input, Γ be the working alphabet of Π, μ its membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$. In the case of P systems with input and *with external output*, the above concepts are introduced in a similar way.

A *recognizer P system* is a P system with input and with external output such that:

- The working alphabet contains two distinguished elements *yes*, *no*.
- All its computations halt.
- 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 external environment associated to the corresponding halting configuration of $\mathcal{C}$.

A decision problem X can be solved in a polynomially uniform way by a family $\mathbf{\Pi} = \{\Pi(n)\}_{n \in \mathbb{N}}$ of P systems of type $\mathcal{F}$ if the following holds:

- There is a deterministic Turing machine M such that, for every $n \in \mathbb{N}$, starting M with the unary representation of n on its input tape, it constructs the P system $\Pi(n)$ in polynomial time in n.
- There is a deterministic Turing machine N that started with an instance $I \in I_X$ with size n on its input tape, it computes a multiset w_I (called the *encoding of* I) over the input alphabet of $\Pi(n)$ in polynomial time in n.
- For every instance $I \in I_X$ with size n, starting $\Pi(n)$ with w_I in its input membrane, every computation of $\Pi(n)$ halts and sends out to the environment *yes* if and only if I is a positive instance of X.

We denote by $\mathbf{PMC}_{\mathcal{F}}$ the set of problems decidable in polynomial time using a polynomially uniform family of P systems of type $\mathcal{F}$.

4 Solving QSAT

In this section, we show that QSAT can be solved in linear time by a polynomially uniform family of recognizer P systems of type $\mathcal{AM}^0_{-d,+mc,+antPri}$.

The QSAT problem is the following one. Given a Boolean formula in conjunctive normal form over the propositional variables $\{x_1,\ldots,x_n\}$. Then the fully (existentially) quantified Boolean formula associated to φ is $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi$, (where Q_n is $\exists$ if n is odd, and it is $\forall$, otherwise). Now, the task is to decide if φ^* is *true*, i.e., to decide if there exists a truth assignment I of the variables $\{x_i \mid 1 \leq i \leq n, i \text{ is odd}\}$ such that each extension I^* of I to the variables $\{x_i \mid 1 \leq i \leq n, i \text{ is even}\}$ satisfies φ.

Next, we construct a recognizer P system of type $\mathcal{AM}^0_{-d,+mc,+antPri}$ to solve QSAT. The construction is a variant of the one occurring in [9] where it is shown that QSAT can be solved in linear time using a family of P systems with membrane creation using dissolution rules. The main difference between the construction in [9] and the one in this paper is that instead of dissolution rules we use annihilation rules to control the computations.

Similarly as in [9], the work of our P systems can be divided into three stages:

- Generation and evaluation stage: Using membrane creation we construct a binary complete tree where the leaves encode all possible truth assignments associated with the formula. The values of the formula corresponding to these truth assignments are obtained in the corresponding leaves. Moreover, the nodes at even (resp. odd) levels from the root are codified by OR gates (respectively, AND gates).
- Checking stage: In this stage the membrane structure corresponds to a Boolean circuit with gates AND and OR. We compute the values of the gates starting with the truth values computed at the leaves towards the root of the circuit which is the output gate.
- Output stage: The system sends out to the environment the answer of the system computed in the previous stages.

The evaluation stage will be the same as in [9], since there no dissolution rules are applied. In the other two stages we will use annihilation rules instead of using membrane dissolution.

Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a Boolean formula in conjunctive normal form over n variables. Then φ can be encoded as a multiset over the alphabet $\{x_{i,j}, y_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$, where $x_{i,j}$ (resp. $y_{i,j}$) represents the fact that x_j (resp. $\neg x_j$) occurs in C_i (notice that since barred objects usually denote antimatters, we cannot use $\bar{x}_{i,j}$ to represent negated variables). Let us denote the above encoding of φ by $cod(\varphi)$. Let us moreover choose an appropriate pairing function $\langle\,,\,\rangle$ from

$\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. We construct a P system $\Pi(\langle n,m\rangle)$ processing the fully quantified formula φ^* associated with φ, when $cod(\varphi)$ is supplied in its input membrane. The family presented here is:

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

where the input alphabet is $\Sigma(\langle n,m\rangle) = \{x_{i,j}, y_{i,j} \mid 1 \le i \le m, 1 \le j \le n\}$, the input membrane is $i(\langle n,m\rangle) = < t, \vee >$, 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:

$$\begin{aligned}\Gamma(\langle n,m\rangle) = \Sigma(\langle n,m\rangle)&\\ &\cup \{z_{j,c} \mid j \in \{0,\dots n\}, c \in \{\wedge,\vee\}\}\\ &\cup \{z_{j,c,l} \mid j \in \{0,\dots,n-1\}, c \in \{\wedge,\vee\}, l \in \{t,f\}\}\\ &\cup \{x_{i,j}, y_{i,j}, x_{i,j,l}, y_{i,j,l} \mid j \in \{1,\dots,n\}, i \in \{1,\dots,m\}, l \in \{t,f\}\}\\ &\cup \{r_i, \overline{r}_i, r_{i,t}, r_{i,f} \mid i \in \{1,\dots,m\}\}\\ &\cup \{p_i, q_i, s_i, t_i, u_i, v_i \mid i \in \{1,2,3\}\} \cup \{\overline{q}_2, \overline{p}_3, \overline{t}_2, \overline{s}_3, \overline{u}_2, \overline{v}_3\}\\ &\cup \{a_i \mid i \in \{0,\dots,n-1\}\} \cup \{b_{i,l} \mid i \in \{1,\dots,n-1\}, l \in \{t,f\}\}\\ &\cup \{c_{i,j} \mid i \in \{1,\dots,n-1\}, j \in \{1,\dots,5(n-i+1)\}\}\\ &\cup \{yes, no, yes_\vee, no_\vee, yes_\wedge, no_\wedge, \overline{yes}_\wedge, \overline{no}_\vee\}.\end{aligned}$$

- The set of labels: $H(\langle n,m\rangle) = \{< l, c > \mid l \in \{t,f\}, c \in \{\wedge,\vee\}\} \cup \{s\}$.
- Initial membrane structure: $\mu = [\,[\,]_{<t,\vee>}\,]_s$.
- Initial multiset: $w_s = \emptyset$, $w_{<t,\vee>} = \{a_0\, z_{0,\wedge,t}\, z_{0,\wedge,f}\}$.
- Input membrane: $[\,]_{<t,\vee>}$.
- 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. $$\left.\begin{array}{l}[z_{j,c} \to z_{j,c,t}\, z_{j,c,f}]_{<l,\overline{c}>}\\ [z_{j,c,l} \to [z_{j+1,\overline{c}}]_{<l,c>}]_{<l',\overline{c}>}\end{array}\right\}\text{ for }\begin{array}{ll} l,l' \in \{t,f\}, & c \in \{\vee,\wedge\},\\ j \in \{0,\dots,n-1\}.\end{array}$$

With these rules the P system creates a nested membrane structure with 2^n innermost cells each of which corresponding to a truth assignment of the variables of the input formula. At the first step, 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 the second step these objects 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. $$\left.\begin{array}{l}[x_{i,j} \to x_{i,j,t}\, x_{i,j,f}]_{<l,c>}\\ [y_{i,j} \to y_{i,j,t}\, y_{i,j,f}]_{<l,c>}\\ [r_i \to r_{i,t}\, r_{i,f}]_{<l,c>}\end{array}\right\}\text{ for }\begin{array}{ll} l \in \{t,f\} & i \in \{1,\dots,m\},\\ c \in \{\vee,\wedge\} & j \in \{1,\dots,n\}.\end{array}$$

These rules duplicate the objects representing the formula. One copy corresponds to the case when the variable is assigned *true*, the other copy corresponds to the case when it is assigned *false*. The objects r_i are also duplicated $(r_{i,t},\ r_{i,f})$ in order to keep track of those clauses that evaluate true on the previous assignments to the variables.

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

Using these rules the P system can evaluate which clauses are *true* under the possible (*true* or *false*) truth assignments of the corresponding variable.

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

In order to analyse the next variable the second subscript of the objects $x_{i,j,l}$ and $y_{i,j,l}$ 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. $[z_{n,c} \rightarrow \overline{r}_1 \dots \overline{r}_m\, p_1\, q_1]_{<l,\overline{c}>}$ for $l \in \{t,f\}$ and $c \in \{\vee,\wedge\}$.
After the evaluation stage, these rules introduce antimatters $\overline{z}_i$, $i \in \{1,\dots,m\}$, in the inner membranes. These antimatters will be used to check if there is a clause that is not satisfied by the corresponding truth assignment.

6. $$\left.\begin{array}{l} [r_i\, \overline{r}_i \rightarrow \lambda]_{<l,c>} \\ [\overline{r}_i \rightarrow \overline{q}_2]_{<l,c>} \end{array}\right\} \text{for } \begin{array}{ll} l \in \{t,f\}, & i \in \{1,\dots,m\}, \\ c \in \{\vee,\wedge\} & \end{array}$$

If an antimatter is not annihilated by the first rule, i.e., there is a clause that is not satisfied by the corresponding truth assignment, then this antimatter introduces the antimatter $\overline{q}_2$.

7. $$\left.\begin{array}{l} [q_1 \rightarrow q_2]_{<l,c>} \\ [p_1 \rightarrow p_2]_{<l,c>} \\ [q_2\overline{q}_2 \rightarrow \lambda]_{<l,c>} \\ [p_3\overline{p}_3 \rightarrow \lambda]_{<l,c>} \\ [p_2 \rightarrow p_3]_{<l,c>} \\ [p_3 \rightarrow no]_{<l,c>} \\ [q_2 \rightarrow q_3\overline{p}_3]_{<l,c>} \\ [q_3 \rightarrow yes]_{<l,c>} \end{array}\right\} \text{for } \begin{array}{l} l \in \{t,f\} \\ c \in \{\vee,\wedge\} \end{array}$$

These rules introduce *yes* in an innermost cell with label $< l, c >$ if and only if the antimatter $\overline{q}_2$ is not present in this cell. On the other hand, if $\overline{q}_2$ is in the cell, then object *no* is introduced. Since $\overline{q}_2$ is introduced if and only if the corresponding truth assignment does not satisfy all the clauses of the formula, the appearance of *yes* or *no* in this cell indicates correctly whether the corresponding truth assignment satisfies the formula or not.

8. $$\left.\begin{array}{l} [yes]_{<l,c>} \rightarrow yes_{\overline{c}}\,[\,]_{<l,c>} \\ [no]_{<l,c>} \rightarrow no_{\overline{c}}\,[\,]_{<l,c>} \end{array}\right\} \text{ for } \begin{array}{c} l \in \{t,f\} \\ c \in \{\vee,\wedge\} \end{array}$$

These rules with the rules in groups **9** and **10** below will be used to check whether an appropriate combination of truth assignments according to the quantifiers $\exists$ and $\forall$ are founded or not.

9. $$\left.\begin{array}{l} [yes_\wedge\,\overline{yes}_\wedge \rightarrow \lambda]_{<l,\wedge>} \\ [t_1 \rightarrow t_2]_{<l,\wedge>} \\ [s_1 \rightarrow s_2]_{<l,\wedge>} \\ [t_2 \rightarrow t_3\,\overline{s}_3]_{<l,\wedge>} \\ [s_2 \rightarrow s_3]_{<l,\wedge>} \\ [t_3]_{<l,\wedge>} \rightarrow yes_\vee\,[\,]_{<l,\wedge>} \\ [\overline{yes}_\wedge \rightarrow \overline{t}_2]_{<l,\wedge>} \\ [s_3]_{<l,\wedge>} \rightarrow no_\vee\,[\,]_{<l,\wedge>} \\ [t_2\,\overline{t_2} \rightarrow \lambda]_{<l,\wedge>} \\ [s_3\,\overline{s_3} \rightarrow \lambda]_{<l,\wedge>} \end{array}\right\} \text{ for } l \in \{t,f\}$$

10. $$\left.\begin{array}{l} [no_\vee\,\overline{no}_\vee \rightarrow \lambda]_{<l,\vee>} \\ [u_1 \rightarrow u_2]_{<l,\vee>} \\ [v_1 \rightarrow v_2]_{<l,\vee>} \\ [u_2 \rightarrow u_3\,\overline{v}_3]_{<l,\vee>} \\ [v_2 \rightarrow v_3]_{<l,\vee>} \\ [u_3]_{<l,\vee>} \rightarrow no_\wedge\,[\,]_{<l,\vee>} \\ [\overline{no}_\vee \rightarrow \overline{u}_2]_{<l,\vee>} \\ [v_3]_{<l,\vee>} \rightarrow yes_\wedge\,[\,]_{<l,\vee>} \\ [u_2\,\overline{u_2} \rightarrow \lambda]_{<l,\vee>} \\ [v_3\,\overline{v_3} \rightarrow \lambda]_{<l,\vee>} \end{array}\right\} \text{ for } l \in \{t,f\}$$

11. $$\left.\begin{array}{l} [a_i \rightarrow b_{i+1,t}\,b_{i+1,f}\,c_{i+1,1}]_{<l,c>} \\ b_{i+1,l}\,[\,]_{<l,c>} \rightarrow [a_{i+1}]_{<l,c>} \end{array}\right\} \text{ for } \begin{array}{l} l \in \{t,f\}, \quad i \in \{0,\ldots,n-2\}, \\ c \in \{\vee,\wedge\} \end{array}$$

These rules with the rules in groups **12-14** below will be used to introduce the multisets $s_1t_1\overline{yes}_\wedge^2$ and $u_1v_1\overline{no}_\vee^2$ in the appropriate membranes. These multisets will be used then by rules in groups **9** and **10**, respectively. Since these multisets will be needed at different levels in the membrane structure in different time steps, we need to employ a counter $c_{i,j}$ for the appropriate timing (see also the groups below).

12. $[a_{n-1} \to c_{n-1,1}]_{<l,c>} \}$ for $l \in \{t,f\}, \quad c \in \{\vee,\wedge\}$

13. $[c_{i,j} \to c_{i,j+1}]_{<l,c>} \}$ for $\begin{array}{ll} l \in \{t,f\}, & i \in \{1,\dots,n\}, \\ c \in \{\vee,\wedge\}, & j \in \{1,\dots,5n-5i+4\} \end{array}$

14. $\left.\begin{array}{l} [c_{n-k,5k+5} \to s_1\, t_1\, \overline{yes}_{\wedge}^2]_{<l,\wedge>} \\ [c_{n-k,5k+5} \to u_1\, v_1\, \overline{no}_{\vee}^2]_{<l,\vee>} \end{array}\right\}$ for $\begin{array}{l} l \in \{t,f\} \\ k \in \{0,\dots,n-1\} \end{array}$

15. $[yes_{\wedge}]_s \to yes\,[\,]_s$
$[no_{\wedge}]_s \to no\,[\,]_s$.

These rules are used to send out the computed answer to the environment.

4.1 A Short Overview of the Computation

The initial configuration only has two membranes, the skin and an elementary membrane with label $< t, \vee >$. Labels have two types of information. On the one hand, the first symbol can be t or f, (*true* of *false*) and the second symbol can be $\wedge$ or $\vee$ to denote if the corresponding variable is universally or existentially quantified. Membrane creation rules are applied in parallel in order to obtain a binary tree like structure of membranes enclosed in the skin. In the $2n$-th step of the computation, 2^n elementary membranes are created. One for each possible truth assignment of the variables. The key set of rules for the evaluation of the variables is the set **3**. According to this set of rules, a symbol r_j is produced for each variable such that its truth value makes true the clause C_j.

Each of the 2^n elementary membranes in the configuration after $2n$ steps can be seen as one of the possible truth assignments for the variables and the set of different r_j objects inside represent the set of clauses satisfied by the corresponding truth assignment. In order to check if all the clauses are satisfied, a set with all the antiparticle $\overline{r}_j$ objects is generated in each elementary membrane. If all of these $\overline{r}_j$ objects are annihilated, it means that in this elementary membrane there were all the objects r_j (maybe with multiple copies). This means that the truth assignment associated with the elementary membrane satisfies all the clauses. Otherwise, if any $\overline{r}_j$ is not consumed after the annihilation process, then we conclude that the corresponding assignment does not satisfy the corresponding clause.

A set of technical rules produce an object *yes* or *no* inside each elementary membrane. The target of most of these rules is to control that only one object *yes* or *no* is generated, regardless the possible combination of multiple copies of r_j in the membrane.

Once the objects *yes* and *no* are generated in the elementary membranes, they are sent up in the tree-like membrane structure. When two of these objects arrive to an intermediate membrane, a new object *yes* or *no* is sent up, according to the label of the membrane. Such label encodes the type of quantification (universal or existential) of the corresponding variable. This stage is controlled by rules from the sets **9** and **10**.

Finally, an object *yes* or *no* arrives to the skin and it is sent out to the environment.

Consequently, the family $\mathbf{\Pi}$ solves in linear time the QSAT problem. Since QSAT is a **PSPACE**-complete problem, we have the following result:

Theorem 1. $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d,+mc,+antPri}}$.

5 PSPACE upper bound

In this section we show that $PMC^0_{\mathcal{AM}_{-d,+mc,+antPri}} \subseteq$ **PSPACE**. The proof is similar to the corresponding one in [23] where it is shown that $PMC_{\mathcal{AM}_{+d,+ne}} \subseteq$ **PSPACE** (i.e., polynomially uniform families of P systems with active membranes, with polarizations, with dissolution and nonelementary membrane division rules can solve only problems in **PSPACE**). Nevertheless, there are substantial differences between the two proofs due to the different behaviour of these systems. In [23] it is observed that the multiset content and the polarization (so called, the *state*) of a membrane M after n steps of a P system Π can be obtained by recursively calculating the states of M, its parent, and its children after $n-1$ steps. To achieve that always the same computations are calculated by the recursive calls, a weak determinism on the rules of Π was introduced in [23] (notice that since Π is a recognizer P system, it is confluent and thus it is enough to simulate only one of its computations). Moreover, to distinguish between membranes having same labels, unique indexes were associated to the membranes of a configuration. The index of a new membrane in a configuration is derived from the index of the corresponding membrane in the previous configuration.

In our proof, on the one hand, we do not have to deal with the polarizations of the membranes. On the other hand, we should employ an indexing technique that is different to that occurring in [23] due to the reason that in P systems with membrane creation new membranes are created from objects and not from membranes. The rest of this section is devoted to the proof of the following theorem:

Theorem 2. $\mathbf{PMC}_{\mathcal{AM}^0_{-d,+mc,+antPri}} \subseteq$ ***PSPACE***.

We give an algorithm A with the following properties. Let $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ be a polynomially uniform family of recognizer P systems of type $\mathcal{AM}_{-d,+mc,+antPri}$. Then, for every $n \in \mathbb{N}$ and input multiset m of $\Pi(n)$, A decides using polynomial space in n if $\Pi(n)$ produces *yes* started on input m.

Assume that $\Pi(n) = (\Gamma, H, \mu, W, h_i, R)$. Since $\Pi(n)$ is a recognizer P system, all of its computations yield the same answer. Thus, it is enough to simulate one particular computation of $\Pi(n)$. To this end, we introduce the following weak priorities on the rules other than annihilation rules in R (clearly, by definition, annihilation rules have priority over the rest of the rules). We assume that evolution rules have the highest priority, followed by send-out communication, send-in communication, and membrane creation rules. Similar type of rules have priority over each other as follows. Assume we have two rules r_1 and r_2 of the same type. Then r_1 has priority over r_2 if and only if one of the following conditions holds:

- $r_1 = [a \to \alpha]_i$, $r_2 = [a \to \beta]_i$ and $\alpha < \beta$ (where $<$ is the usual lexicographical order on words),
- $r_1 = a[\ ]_i \to [b]_i$, $r_2 = a[\ ]_j \to [c]_j$ and ($i < j$ or ($i = j$ and $b < c$)),
- $r_1 = [a]_i \to b[\ ]_i$, $r_2 = [a]_i \to c[\ ]_i$ and $b < c$,
- $r_1 = [a \to [\alpha]_j]_i$, $r_2 = [a \to [\beta]_k]_i$ and ($j < k$ or ($j = k$ and $\alpha < \beta$)).

One can see that even with the above priorities, $\Pi(n)$ can have different computations on the same input. Indeed, assume, for example, that $\Pi(n)$ has a configuration which contains a membrane structure $[[\ ]_2\ [\ ]_2]_1$ with an object a in membrane 1. Assume also that $\Pi(n)$ has the rule $r = a[\ ]_2 \to [b]_2$. Then when $\Pi(n)$ applies r, it nondeterministically chooses a membrane with label 1 and sends a into this membrane. It also can bee seen that there is no such nondeterminism concerning the other types of rules. As we will see later, using unique indexes of the membranes having the same labels, we can avoid of this nondeterminism during the simulation.

Next we define these unique indexes. First of all, we assume that different membranes have different labels in the initial configuration. Assume now that $C = C_1, \dots, C_l$ is a computation of $\Pi(n)$. Let $i \in \{1, \dots, l\}$ and M be a membrane in C_i. Let $d(M, C_i)$ denote the depth of M in the membrane structure in C_i. More precisely, if M is the skin, then $d(M, C_i) = 1$; if M is a child of a membrane M', then $d(M, C_i) := d(M', C_i) + 1$. Let moreover $d(C_i) := max\{d(M, C_i) \mid M \text{ is a membrane in } C_i\}$. We inductively define a function f_C that assigns to every membrane M in C_i an index from $((H \cup \mathbb{N})^{i+1})^{d(M,C_i)}$ (i.e., the index of M will be a $d(M, C_i)$-tuple of words with length $i+1$ containing letters from $H \cup \mathbb{N}$). The indexes of the membranes in C_1 are inductively defined as follows. For the skin membrane M with label s, let $F_C(M) := (s1)$. Now let M be a membrane in C_1 and assume that $F_C(M) = (w_1, \dots, w_{d(M,C_1)})$. If M' is a child membrane of M with label h, then $F_C(M') := (h1, w_1, \dots, w_{d(M,C_1)})$. An example of this indexing in the initial configuration can be seen on Fig. 1, where these indexes are written in the lower-right corner of the membranes. Now assume that f_C already assigns

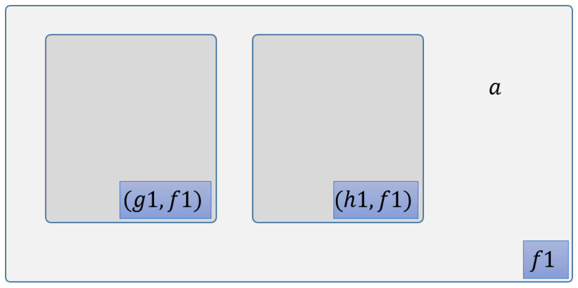

Fig. 1.

an index to every membrane in C_i ($i < l$). Let M be a membrane in C_i and assume that $f_C(M) = (w_1, \ldots, w_{d(M,C_i)})$. If M' is the membrane in C_{i+1} that corresponds to M, then let $f_C(M') = (w_1 1, \ldots, w_{d(M,C_i)} 1)$ (notice that since dissolution and membrane duplication rules are not allowed, every membrane in C_i has a corresponding membrane in C_{i+1}). Finally, let $h \in H$ and assume that $a_1, \ldots, a_k$ are those objects in M (ordered lexicographically) that create membranes with label h in the step from C_i to C_{i+1}. For every $j \in \{1, \ldots, k\}$, let M_j be that membrane which is created from a_j. Then $f_C(M_j) := (h a_j^i j, w_1 1, \ldots, w_{d(M,C_i)} 1)$. An example of this indexing can be seen in Fig. 2, where at the first step a enters to membrane with index $(h1, f1)$ and evolves to b. Then, during the second step, b creates the membrane with index $(gbb1, h111, f111)$. Notice that from this index we can

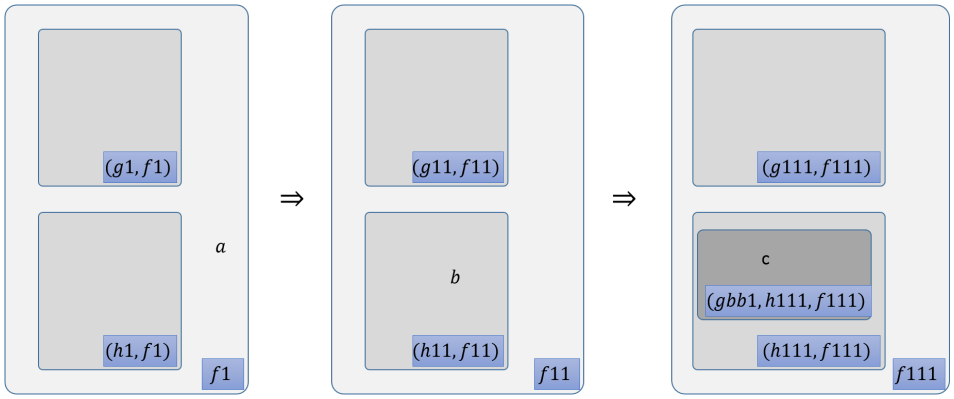

Fig. 2.

decode the following information. The label of the membrane is g, its parent has label h and index $(h111, f111)$, and the membrane was created in the second step of the computation from an object b. In general, the above defined indexes have the following properties:

- For a given initial configuration C_1 and a computation $C = C_1, \ldots, C_l$, the possible indexes of the membranes in C can be effectively enumerated (notice that the maximal number of objects in a membrane can be calculated from the number of objects in C_1 and the number of computation steps);
- For a membrane M with index $(h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j})$,
  - if $k > 1$, then the index of the parent membrane of M is $(h_2 i_{2,1} \ldots i_{2,j}, \ldots, h_k i_{k,1} \ldots i_{k,j})$, and
  - the possible indexes of the children of M can be effectively enumerated;
- For a membrane M with index $(h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j})$ such that $j > 1$, either
  - $i_{1,1} = \ldots = i_{1,j} = 1$ and M occurs already in the initial configuration, or

– $i_{1,1} = \ldots = i_{1,j-1} = a$, for some $a \in \Gamma$, and M is created from a in the $(j-1)$th step of the computation.

Let $C = C_1, \ldots, C_l$ be a computation of $\Pi(n)$ and $j \in \{1, \ldots, l\}$. We introduce an order on the indexes of membranes occurring in C_j and satisfying that $d(M, C_j) = d(M', C_j)$. Assume that M and M' are membranes with these properties and $f_C(M) = (h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j})$, and $f_C(M') = (h'_1 i'_{1,1} \ldots i'_{1,j}, \ldots, h'_k i'_{k,1} \ldots i'_{k,j})$, where $k = d(M, C_j)$. Then $(h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j}) \leq (h'_1 i'_{1,1} \ldots i'_{1,j}, \ldots, h'_k i'_{k,1} \ldots i'_{k,j})$ if and only if $h_1 i_{1,1} \ldots i_{1,j} \leq h'_1 i'_{1,1} \ldots i'_{1,j}$, where $\leq$ is the usual lexicographical order on words assuming that, for every object $a \in \Gamma$ and number $n \in \mathbb{N}$, $a < n$.

Let $C = C_1, \ldots, C_l$ be a halting computation of $\Pi(n)$ such that, for every $i \in \{1, \ldots, l\}$, C_i has the following property. Assume that there is a membrane M with label h in C_i and there are more than one membranes with label g in M. Assume also that there is a rule $r = a[\ ]_g \to [b]_g$ in R. Then C_{i+1} is that configuration of $\Pi(n)$ where the objects a in M are sent by the rule r to that membrane with label g which has a smaller index by the above defined order on the indexes. We will simulate this particular computation C by recursively calculating the multiset content of membranes in C. This is done using a function called Content. Content gets as parameters an index of a membrane M and a number j and returns with the multiset content of M in C_j (i.e., the content of M after $j-1$ computation steps). The basic strategy of the computation, roughly, is the following. First we try to compute the content of M and the content of its parent M' in C_{j-1}. If M' does not exist in C_{j-1}, then M also does not exist and we can return *nil* showing that the content of M in C_j is undefined. If only M does not exist in C_{j-1}, we check whether it was created in the step from C_{j-1} to C_j. If no, then we return *nil*, otherwise we return the newly created content of M. If both M and M' exist in C_{j-1}, then we calculate the content of M in C_j using the contents of M and M' in C_{j-1} and by calculating the contents of the children of M in C_{j-1}.

For the better readability, in the algorithms defined below we will refer to the annihilation (resp. evolution, send-out communication, send-in communication, and membrane creation) rules as *ann* (resp. *evo*, *in_com*, *out_com*, and *cre*).

```
1. function Content($(h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j}), j$)
   // We calculate the multiset content of a membrane $M$ with index
   $(h_1 i_{1,1} \ldots i_{1,j}, \ldots, h_k i_{k,1} \ldots i_{k,j})$ in $C_j$;
2. if $j = 1$ then
3.    if $i_{1,1} = 1, \ldots, i_{k,1} = 1$ AND there is a membrane structure $\mu =$
      $[\,[\,[\,]_{h_1} \ldots]_{h_{k-1}}\,]_{h_k}$ in $C_1$ then
4.       return the multiset content of the inner membrane in $\mu$
5.    else return nil
6.    end if;
7.    exit
```

```
8.  end if;
    // If $j = 1$ and the index corresponds to a membrane in $C_1$, then return the content
    of this membrane, and return $nil$, otherwise;
9.  $S \leftarrow$ Content$((h_1 i_{1,1} \dots i_{1,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}), j-1)$;
    // If $j > 1$, then we recursively calculate the content of $M$ in $C_{j-1}$;
10. $S_p \leftarrow \varnothing$; $S' \leftarrow \varnothing$; $S_c \leftarrow \varnothing$; $X' \leftarrow \varnothing$;
11. if $h_1$ is not the label of the skin membrane then
12.     $S_p \leftarrow$ Content$((h_2 i_{2,1} \dots i_{2,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}), j-1)$;
        // If $M$ is not the skin, then we calculate the content of the parent $M'$ of $M$ in
        $C_{j-1}$;
13.     if $S_p = nil$ then return $nil$; exit
        // If the parent $M'$ of $M$ does not exist in $C_{j-1}$, then $M$ cannot exist in $C_j$;
14.     else
15.         TryRules$(h_2, ann, S_p, X', X')$;
16.         TryRules$(h_2, evo, S_p, X', X')$;
17.         TryRules$(h_2, out\_com, S_p, X', X')$;
            // We remove from $S_p$ those objects that do not contribute to the content of $M$
            in $C_j$ by applying rules $ann$, $evo$, and $out\_com$ to the content of $M'$ in $C_{j-1}$;
18.         for all possible index $(h'_1 i'_{1,1} \dots i'_{1,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1})$ such
            that $(h'_1 i'_{1,1} \dots i'_{1,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}) < (h_1 i_{1,1} \dots i_{1,j-1}, \dots,$
            $h_k i_{k,1} \dots i_{k,j-1})$
19.             $S_c \leftarrow$ Content$((h'_1 i'_{1,1} \dots i'_{1,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}), j-1)$;
20.             if $S_c \neq nil$ then
21.                 TryRules$(h'_1, in\_com, X', X', S_p)$
22.             end if
23.         end for
        // We remove those objects from the content of $M'$ that are sent to child mem-
        branes other than $M$;
24.     end if
25. end if
26. if $S \neq nil$ then
27.     TryRules$(h_1, ann, S, X', X')$;
28.     TryRules$(h_1, evo, S, S', X')$;
29.     TryRules$(h_1, out\_com, S, X', X')$;
        //We apply rules $ann$, $evo$, and $out\_com$ to the content $S$ of $M$ in $C_{j-1}$;
30.     TryRules$(h_1, in\_com, S_p, S', X')$
        // We send objects from the parent $M'$ to the children $M$ by applying $in\_com$
        rules;
31.     CommWithChildren$((h_1 i_{1,1} \dots i_{1,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}), j-1, S, S')$;
        //We calculate the interactions between $M$ and its children in $C_{j-1}$;
32.     $S' \leftarrow S \cup S'$;
        // We calculate the content of $M$ in $C_j$; here $S$ contains those objects that
        were not involved by any rule; $S'$ contains the results of the applicable rules;
33.     return $S'$; exit
34. end if
```

```
35. if $S = nil$ then
36.   CommWithChildren($(h_2 i_{2,1} \dots i_{2,j-1}, \dots, h_k i_{k,1} \dots i_{k,j-1}), j-1, S_p, X'$);
      // If $M$ does not exists in $C_{j-1}$, then we examine if it can be created in $M'$ in the
      step from $C_{j-1}$ to $C_j$; first we remove those objects from the content of $M'$ that
      are sent to its children during the step form $C_{j-1}$ to $C_j$;
37.   if $a_1 \dots a_t \subseteq S_p$ $(a_1 \leq \dots \leq a_t)$ such that $i_{1,1} = \dots = i_{1,j-1} = a_t$ AND
      $i_{1,j} = t$ AND $[a_t \to [\,v\,]_{h_1}]_{h_2} \in R$ then
38.     $S' \leftarrow v$;
        // If $a_1, \dots, a_t$ occur in $S_p$ and $M$ can be created in $M'$ by the rule $[a_t \to$
        $[\,v\,]_{h_1}]_{h_2}$, then the content of $M$ in $C_j$ is $v$;
39.     return $S'$
40.   else
41.     return $nil$
42.   end if
43. end if
```

Next we define the procedure TryRules which have five parameters. The first one is a label of the membrane, the next one is a type of rules, and the last three parameters are those sets of objects that are involved by the application of the corresponding type of rules.

```
 1. procedure TryRules($g, type, X, Y, Z$)
 2. case $type$ of
 3.   $ann$: for each rule $[\,a\overline{a} \to \lambda\,]_g$ do
 4.     remove every pair $a, \overline{a}$ from $X$
 5.   end for
 6.   $evo$: for each rule $[\,a \to \alpha\,]_g$ do
 7.     remove every occurrence of $a$ from $X$;
 8.     add to $Y$ the same number of multiset represented by $\alpha$
 9.   end for
10.   $in\_com$: for each rule $a[\;]_g \to [\,b\,]_g$ do
11.     remove every occurrence of $a$ from $Z$;
12.     add to $Y$ the same number of objects $b$
13.   end for
14.   $out\_com$: for each rule $[\,a\,]_g \to b[\;]_g$ do
15.     remove every occurrence of $a$ from $X$;
16.     add to $Z$ the same number of objects $b$
17.   end for
18.   $cre$: for each rule $[\,a \to [\;]_h\,]_g$ do
19.     remove every occurrence of $a$ from $X$
20.   end for
21. end case
```

Now, we define the procedure CommWithChildren which calculates the communications between a membrane and its children. This procedure has four parameters. The second parameter is a number j which determines which step of the computation is considered. The first parameter is an index of a membrane in C_j. The last two parameters are those sets of objects that are involved by the communications between this membrane and its children in the step from C_j to C_{j+1}.

```
 1. procedure CommWithChildren($(h_1 i_{1,1} \dots i_{1,j}, \dots, h_k i_{k,1} \dots i_{k,j}), j, X, Y$)
 2. for each $g l_{1,1} \dots l_{1,j}$, where $g \in H, l_{1,1}, \dots, l_{1,j} \in H \cup \mathbb{N}$ do
 3.   $S_c \leftarrow Content((g l_{1,1} \dots l_{1,j}, h_1 i_{1,1} \dots i_{1,j}, \dots, h_k i_{k,1} \dots i_{k,j}), j)$;
 4.   if $S_c \neq nil$ then
 5.     $Y' \leftarrow \varnothing$;
 6.     TryRules($g, ann, S_c, Y', Y'$);
 7.     TryRules($g, evo, S_c, Y', Y'$);
 8.     TryRules($g, out\_com, S_c, Y', Y$);
 9.     TryRules($g, in\_com, Y', Y', X$);
        // We apply rules of type $ann$ and $evo$ to keep the computation deterministic;
        membrane creation rules are skipped as they do not contribute to the content
        of the parent membrane stored in $X$; in-communication rules involve only the
        content of the parent membrane;
10.   end if
11. end for
```

Finally, we present the procedure A to decide if $\Pi(n)$ sends out to the environment *yes* on a given input multiset m. We assume without loss of generality that those rules that send out to the environment *yes* (resp. *no*) have the form $[\,yes\,]_s \rightarrow yes$ (resp. $[\,yes\,]_s \rightarrow yes$), where s is the label of the skin membrane.

```
 1. procedure A($\Pi(n)$)
    // $\Pi(n) = (\Gamma, H, \mu, W, h_i, R)$ is a recognizer P systems of type $\mathcal{AM}_{-d,+mc,+antPri}$
    with input multiset $m$
 2. $s \leftarrow$ the label of the skin in $\mu$;
 3. $S \leftarrow \varnothing$;
 4. for each $j = 1, 2, \dots$ do
 5.   $S \leftarrow$ Content($(s i_1 \dots i_j), j$) where $i_1 = 1, \dots, i_j = 1$;
 6.   if $yes \in S$ and there is a rule $[\,yes\,]_s \rightarrow yes$ then output: $yes$; exit
 7.   end if
 8.   if $no \in S$ and there is a rule $[\,no\,]_s \rightarrow no$ then output: $no$; exit
 9.   end if
10. end for
```

First we show that A halts on $\Pi(n)$ and m. Since Content recursively calls itself with a decreasing second parameter and Content with second parameter 1 exits after finite steps, we can conclude that Content always exits after finite steps. Moreover, since $\Pi(n)$ is a recognizer P system, it halts in l steps, for an appropriate number l. Thus the multiset content of the skin of Π should contain *yes* or *no* after at most $l - 1$ steps. Therefore, $l + 1$ is the highest number that occurs as a second parameter in the calls of Content in A. This means that A stops after a finite number of steps.

Next we discuss the space complexity of A. Let $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ be a polynomially uniform family of P systems of type $\mathcal{AM}_{-d,+mc,+antPri}$. By definition, there is a polynomial $p(n)$ such that the size of the initial configuration of $\Pi(n)$ containing an encoding of a formula in its input membrane is upper bounded by $p(n)$. Moreover, there is a polynomial $t(n)$ such that the running time of $\Pi(n)$ is upper bounded by $t(n)$.

Let $C = C_1, \dots, C_l$ be a halting computation of $\Pi(n)$, for some $l \leq t(n)$, and M be a membrane in C_i ($i \in \{1, \dots, l\}$). Then the index $w = f_C(M)$ contains at most $k + i - 1$ components, where $k = d(C_1)$. Clearly, k is upper bounded by $p(n)$. Moreover, every component of w is a word with length at most $t(n) + 2$. It follows then that w contains at most $(p(n)+t(n)-1)\cdot(t(n)+2)$ letters. Clearly, for every $i \in \{1, \dots, l\}$, the size of C_i is at most $p(n)^{O(t(n))}$ (the size of a configuration is the sum of the number of objects and membranes in the configuration). Thus, every letter in w that is contained in $\mathbb{N}$ is at most $p(n)^{k\cdot t(n)}$, for some appropriate constant k. Therefore, storing a letter of a word in w needs at most $\log(p(n)^{k\cdot t(n)}) = O(nt(n))$ bits (notice that the first letters of the words in w are labels and the number of different labels is bounded by $p(n)$). This implies that the index w can be stored using at most $O((p(n) + t(n)) \cdot t(n) \cdot nt(n))$ bits, i.e., the number of necessary bits is polynomial in n. Therefore, on every level of the recursion in the function Content, the number of bits that is used to store the parameters is upper bounded by an appropriate polynomial. Moreover, the depth of the recursion in Content is bounded by the poynomial $t(n)$. It follows, that the space complexity of A is bounded by a polinomial too.

6 Conclusions and Future Work

In this paper, we have proved that the family of P systems with membrane creation and annihilation rules characterizes the complexity class **PSPACE**. In [5] it has been proved that P systems with active membranes without polarizations, without dissolution and with division of elementary and non-elementary membranes endowed with antimatter and annihilation rules can solve **NP**-complete problems. It is an interesting research topic to explore the exact computational power of these systems. It seems that these systems can only solve problems in **PSPACE**. On the other hand, solving a **PSPACE**-complete problem with these systems seems to be a challenging task.

Acknowledgements

MAGN acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

1. Alhazov, A., Aman, B., Freund, R.: P systems with anti-matter. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 66–85. Springer (2014)
2. Alhazov, A., Aman, B., Freund, R., Păun, Gh.: Matter and anti-matter in membrane systems. In: DCFS 2014. Lecture Notes in Computer Science, vol. 8614, pp. 65–76. Springer (2014)
3. Berman, L.: The complexity of logical theories. Theoretical Computer Science 11, 71–77 (1980)
4. Blizard, W.D.: Negative membership. Notre Dame Journal of Formal Logic 31(3), 346–368 (1990)
5. Díaz-Pernil, D., Peña-Cantillana, F., Alhazov, A., Freund, R., Gutiérrez-Naranjo, M.A.: Antimatter as a frontier of tractability in membrane computing. Fundamenta Informaticae 134, 83–96 (2014)
6. Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 3850. Springer, Berlin Heidelberg (2006)
7. Gott, J.: Time Travel in Einstein's Universe: The Physical Possibilities of Travel Through Time. Houghton Mifflin (2001)
8. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund et al. [6], pp. 224–240
9. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution for QSAT with membrane creation. In: Freund et al. [6], pp. 241–252
10. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution of subset sum problem by using membrane creation. In: Mira, J., Álvarez, J.R. (eds.) IWINAC (1). Lecture Notes in Computer Science, vol. 3561, pp. 258–267. Springer, Berlin Heidelberg (2005)
11. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. Theoretical Computer Science 371(1-2), 54–61 (2007)
12. Ito, M., Martín-Vide, C., Păun, G.: A characterization of parikh sets of ET0L languages in terms of P systems. In: Ito, M., Păun, G., Yu, S. (eds.) Words, Semigroups, and Transductions - Festschrift in Honor of Gabriel Thierrin. pp. 239–253. World Scientific (2001)
13. Loeb, D.: Sets with a negative number of elements. Advances in Mathematics 91, 64–74 (1992)
14. Luisi, P.: The chemical implementation of autopoiesis. In: Fleischaker, G., Colonna, S., Luisi, P. (eds.) Self-Production of Supramolecular Structures, NATO ASI Series, vol. 446, pp. 179–197. Springer Netherlands (1994)
15. Metta, V.P., Krithivasan, K., Garg, D.: Computability of spiking nueral P systems with anti-spikes. New Mathematics and Natural Computation (NMNC) 08(03), 283–295 (2012)
16. Mutyam, M., Krithivasan, K.: P systems with membrane creation: Universality and efficiency. In: Margenstern, M., Rogozhin, Y. (eds.) MCU. Lecture Notes in Computer Science, vol. 2055, pp. 276–287. Springer (2001)

17. Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. International Journal of Computers, Communications & Control IV(3), 273–282 (September 2009)
18. Pérez-Jiménez, M.J.: An approach to computational complexity in membrane computing. In: Mauri, G., Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 3365, pp. 85–109. Springer (2004)
19. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 302 – 336. Oxford University Press, Oxford, England (2010)
20. Păun, Gh.: Some quick research topics., In these proceedings.
21. Păun, G.: Four (somewhat nonstandard) research topics. In: Maccías-Ramos, L.F., del Amor, M.A.M., Păun, G., Riscos-Núñez, A., Valencia-Cabrera, L. (eds.) Twelfth Brainstorming Week on Membrane Computing. pp. 305–309. Fénix Editora, Sevilla, Spain (2014)
22. Song, T., Jiang, Y., Shi, X., Zeng, X.: Small universal spiking neural P systems with anti-spikes. Journal of Computational and Theoretical Nanoscience 10(4), 999–1006 (2013)
23. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. J. Comput. Syst. Sci. 73(1), 137–152 (2007)
24. Tan, G., Song, T., Chen, Z., Zeng, X.: Spiking neural P systems with anti-spikes and without annihilating priority working in a 'flip-flop' way. International Journal of Computing Science and Mathematics 4(2), 152–162 (Jul 2013)