Technical Report No. 2009/1

Institute of Computer Science, Silesian University, 74601 Opava, Czech Republic

On the power of computing with proteins on membranes

Petr Sosík[1,2], Andrei Păun[1,3,4], Alfonso Rodríguez-Patón[1], and David Pérez[1]

[1] Departamento de Inteligencia Artificial, Facultad de Informática
Universidad Politécnica de Madrid, Campus de Montegancedo s/n
Boadilla del Monte, 28660 Madrid, Spain
{psosik,apaun,arpaton,dperez}@fi.upm.es
[2] Institute of Computer Science, Silesian University, 74601 Opava, Czech Republic
[3] Department of Computer Science/IfM, Louisiana Tech University, P.O. Box 10348, Ruston, LA 71272, USA
[4] Bioinformatics Department, National Institute of Research and Development for Biological Sciences, Splaiul Independenţei, Nr. 96, Sector 6, Bucharest, Romania

Abstract. P systems with proteins on membranes are inspired closely by switching protein channels. This model of membrane computing using membrane division has been previously shown to solve an NP-complete problem in polynomial time. In this paper we characterize the class of problems solvable by these P systems in polynomial time and we show that it equals **PSPACE**. Therefore, these P systems are computationally equivalent (up to a polynomial time reduction) to the alternating Turing machine or the PRAM computer. The proof technique we employ reveals also some interesting trade-offs between certain P system properties, as antiport rules, membrane labeling by polarization or the presence of proteins.

1 Introduction

We continue the work on P systems with proteins on membranes, a model combining membrane systems and brane calculi as introduced in [7]. We consider a rather restrictive case, where the "main" information to process is encoded in the multisets from the regions of a P system, but these objects evolve under the control of a bounded number of proteins placed on membranes. Also, the rules we use are very restrictive: move objects across membranes, under the control of membrane proteins, changing or not the objects and/or the proteins during these operations. In some sense, we have an extension of symport/antiport rules

[5], with the mentioning that we always use minimal rules, dealing with only one protein, one object inside the region and/or one object outside of it.

The motivation came from the observation by several authors recently that the maximal parallelism way of processing different species of molecules in the membrane structure is not very close to reality, thus we are considering a model that is limiting the parallelism through the modeling of the trans-membrane proteins (protein channels) observed in nature. A second motivation comes from the brane calculi in which many rules act at the level of the membrane (unlike rules which act within the region enclosed by the membrane). In brane calculi introduced in [3], one works only with objects – called proteins – placed on membranes, while the evolution is based on membrane handling operations, such as exocytosis, phagocytosis, etc. In the membrane computing area we have rules associated with each region defined by a membrane, and in the recent years the rules in membrane computing have been considered mainly to work on symbol objects rather than other structures such as strings. The extension considered in [7] and in [8] was to have both types of rules (both at the level of the region delimited by membranes and also at the level of membrane controlled by a protein). The reason for considering both extensions was that in biology, many reactions taking place in the compartments of living cells are controlled/catalysed by the proteins embedded in the membranes bilayer. For instance, it is estimated that in the animal cells, the proteins constitute about 50% of the mass of the membranes, the rest being lipids and small amounts of carbohydrates. There are several types of such proteins embedded in the membrane of the cell; one simple classification places these proteins into two classes, that of integral proteins (these molecules can "work" in both inside the membrane as well as also in the region outside the membrane), and that of peripheral proteins (macromolecules that can only work in one region of the cell) – see [1].

In this paper we show that P systems with proteins on membranes can solve in polynomial time exactly the class of problems **PSPACE**. Mathematically, this property can be expressed as

$$M\text{-PTIME} = M\text{-NPTIME} = \mathbf{PSPACE}, \tag{1}$$

where M-(N)PTIME is the class of problems solved in polynomial time by a (non-) deterministic machine M. (In our case, the machine M will be a P system with proteins on membranes.) This relation is also known as the *Parallel Computation Thesis* [12]. Computational devices with this property form the so-called *second machine class*. Another members of this class are the alternating Turing machine, SIMDAG (also known as SIMD PRAM) and other standard parallel computer models [12].

The rest of the paper is organized as follows: after introducing basic concepts used throughout the paper in Section 2, we show in Section 3 that the P systems with proteins on membranes can solve the problem QSAT in linear time. Then in Section 4 we show that such a P system can be simulated with a conventional computer (and hence also with Turing machine) in a polynomial space. Section 5 concludes the paper and mentions also some open problems.

2 Definitions

We will start by giving some preliminary notations and definitions which are standard in the area of membrane systems. The reader is referred to [4, 9] for an introduction and overview of membrane systems, and to [13] for the most recent information. The membranes delimit *regions* precisely identified by the membranes. In these regions we place *objects* — elements of the set O. Several copies of the same object can be present in a region, so we work with *multisets* of objects. For a multiset M we denote by $|M|_a$ the multiplicity of objects a in M. A multiset M with the underlying set O can be represented by a string $x \in O^*$ (by O^* we denote the free monoid generated by O with respect to the concatenation and the identity λ) such that the number of occurrences of $a \in O$ in x represents the value $|M|_a$.

In the P systems which we consider below, we use two types of objects, *proteins* and usual *objects*; the former are placed **on** the membranes, the latter are placed **in** the regions delimited by membranes. The fact that a protein p is on a membrane (with label) i is written in the form $[_i p|\ ]_i$. Both the regions of a membrane structure and the membranes can contain multisets of objects and of proteins, respectively.

We consider the types of rules introduced in [7]. In all of these rules, a, b, c, d are objects, p is a protein, and i is a label ("res" stands for "restricted"):

Type	Rule	Effect
1res	$[_i p\|a]_i \to [_i p\|b]_i$	
	$a[_i p\|\ ]_i \to b[_i p\|\ ]_i$	modify an object, but not move
2res	$[_i p\|a]_i \to a[_i p\|\ ]_i$	
	$a[_i p\|\ ]_i \to [_i p\|a]_i$	move an object, but not modify
3res	$[_i p\|a]_i \to b[_i p\|\ ]_i$	
	$a[_i p\|\ ]_i \to [_i p\|b]_i$	modify and move one object
4res	$a[_i p\|b]_i \to b[_i p\|a]_i$	interchange two objects
5res	$a[_i p\|b]_i \to c[_i p\|d]_i$	interchange and modify two objects

In all cases above, the protein is not changed, it plays the role of a catalyst, just assisting the evolution of objects. A generalization is to allow rules of the forms below (now, "cp" means "change protein"):

Type	Rule	Effect (besides changing also the protein)
1cp	$[_i p\|a]_i \to [_i p'\|b]_i$	
	$a[_i p\|\ ]_i \to b[_i p'\|\ ]_i$	modify an object, but not move
2cp	$[_i p\|a]_i \to a[_i p'\|\ ]_i$	
	$a[_i p\|\ ]_i \to [_i p'\|a]_i$	move an object, but not modify
3cp	$[_i p\|a]_i \to b[_i p'\|\ ]_i$	
	$a[_i p\|\ ]_i \to [_i p'\|b]_i$	modify and move one object
4cp	$a[_i p\|b]_i \to b[_i p'\|a]_i$	interchange two objects
5cp	$a[_i p\|b]_i \to c[_i p'\|d]_i$	interchange and modify two objects

where p, p' are two proteins (possibly equal, and then we have rules of type *res*).

An intermediate case can be that of changing proteins, but in a restricted manner, by allowing at most two states for each protein, $p, \bar{p}$, and the rules either as in the first table (without changing the protein), or changing from p to $\bar{p}$ and back (like in the case of bistable catalysts). Rules with such flip-flop proteins are denoted by $nff, n = 1, 2, 3, 4, 5$ (note that in this case we allow both rules which do not change the protein and rules which switch from p to $\bar{p}$ and back).

Both in the case of rules of type *ff* and of type *cp* we can ask that the proteins are always moved in their complementary state (from p into $\bar{p}$ and vice versa). Such rules are said to be of *pure ff* or *cp* type, and we indicate the use of pure *ff* or *cp* rules by writing *ffp* and *cpp*, respectively.

To *divide a membrane*, we use the following type of rule, where p, p', p'' are proteins (possible equal): $[_i p|\,]_i \rightarrow [_i p'|\,]_i [_i p''|\,]_i$

The membrane i is assumed not to have any polarization and it can be non-elementary. The rule doesn't change the membrane label i and instead of one membrane, at next step, will have two membranes with the same label i and the same contents replicated from the original membrane: objects and/or other membranes (although the rule specifies only the proteins involved).

Definition 1. *A* P system with proteins on membranes and membrane division *(in the sequel simply P system, if not stated otherwise) is a system of the form* $\Pi = (O, P, \mu, w_1/z_1, \ldots, w_m/z_m, E, R_1, \ldots, R_m, i_o)$, *where*

m *is the degree of the system (the number of membranes),*
O *is the set of objects,* P *is the set of proteins (with* $O \cap P = \emptyset$*),*
μ *is the membrane structure,*
$w_1, \ldots, w_m$ *are the (strings representing the) multisets of objects present in the* m *regions of the membrane structure* μ*,*
$z_1, \ldots, z_m$ *are the multisets of proteins present on the* m *membranes of* μ*,*
$E \subseteq O$ *is the set of objects present in the environment (in an arbitrarily large number of copies each),*
$R_1, \ldots, R_m$ *are finite sets of rules associated with the* m *membranes of* μ*, and*
i_o *is the label of the output membrane.*

The rules are used in the non-deterministic maximally parallel way: in each step, a maximal multiset of rules is used, that is, no rule is applicable to the objects and the proteins which remain unused by the chosen multiset. At each step we have the condition that each object and each protein can be involved in the application of at most one rule, but the membranes are not considered as involved in the rule applications except the division rules, hence the same membrane can appear in any number of rules of types 1–5 at the same time. By halting computation we understand a sequence of configurations that ends with a halting configuration (there is no rule that can be applied considering the objects and proteins present at that moment in the system). With a halting computation we associate a result, in the form of the multiplicity of objects present in region i_o at the moment when the system halts. We denote by $N(\Pi)$ the set of numbers computed in this way by a given system Π. We denote, in the usual way, by

$NOP_m(pro_r;$*list-of-types-of-rules*$)$ the family of sets of numbers $N(\Pi)$ generated by systems Π with at most m membranes, using rules as specified in the list-of-types-of-rules, and with at most r proteins present on a membrane. When parameters m or r are not bounded, we use $*$ as a subscript.

Example: Consider the P system $\Pi = (O, P, \mu, w_0/z_0, w_1/z_1, E, R_0, R_1, i_0)$, where

$$\begin{aligned}
O &= \{a_1, \ldots, a_n\} \\
P &= \{p, q\} \\
\mu &= [_0[_1]_1]_0 \\
w_0 &= z_0 = E = \emptyset \\
w_1 &= \{a_1\} \\
z_1 &= \{p\} \\
R_0 &= \emptyset \\
R_1 &= \{[_1p|\ ]_1 \to [_1q|\ ]_1[_1q|\ ]_1, [_1q|a_n]_1 \to a_n[_1q|\ ]_1\} \\
&\quad \cup \{[_1q|a_i]_1 \to [_1p|a_{i+1}]_1 \mid 1 \leq i \leq n-1\} \\
i_0 &= 0
\end{aligned}$$

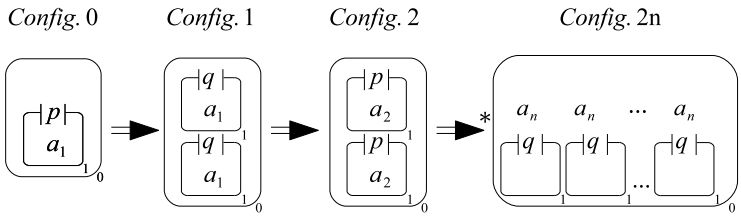

Fig. 1. An example of a P system with proteins on membranes.

In its initial configuration the system contains two membranes and one object. In every odd step all the membranes labelled 1 are divided and their membrane proteins are changed from p to q. In every even step the proteins change back from q to p, and objects a_i in the membranes evolve to a_{i+1}, for $1 \leq i \leq n-1$. Therefore, every two steps the number of membranes labelled 1 is doubled. In $2n$-th step the objects a_n are expelled to the membrane labelled 0, which is the output membrane, and the systems halts. The computation of the system is illustrated in Fig. 1. Therefore, we can write that $N(\Pi) = \{2^n \mid n \in \mathbb{N}\}$.

Several computational universality results are known to hold for P systems with proteins on membranes [8, 7], from which we recall only two:

$$NOP_1(pro_2; 2cpp) = NRE,$$
$$NOP1(pro_*; 3ffp) = NRE,$$

where *NRE* is the class of all recursively enumerable sets of non-negative integers. In this paper, however, we focus on P systems working in accepting mode, described in the next section, which can solve decision problems.

2.1 Families of membrane systems

Most of the membrane computing models are universal, i.e., they allow for a construction of a universal machine capable of solving any Turing-computable problem. However, when we try to employ the massive parallelism of P systems for effective solutions to intractable problems, the concept of one universal P systems solving all the instances of the problem is rather restrictive. The effective use of parallelism can be restricted by the particular structure of such a P system. For instance, the depth of the structure is fixed during the computation in most P system models. But for an effective parallel solution to various instances, various depths of the membrane structure might be needed.

Therefore, to attack intractable problems, we frequently use families of P systems instead of a single P system. Generally, given a computational problem X, each machine M_n of the family $\mathcal{M} = (M_0, M_1, \ldots)$ is able to solve the instances of X of size n. We denote by $|x_i|$ the size of an instance x_i of a problem X. In the usual representation x_i, $i = 1, 2, \ldots$, are words over a fixed finite alphabet and $|x_i|$ is the length of x_i. The following definition is due to [6].

Definition 2. *Let $\mathcal{D}$ be a class of P systems and let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a total function. The* class of problems solved by uniform families of P systems of type $\mathcal{D}$ in time f, *denoted by* $\mathbf{MC}_{\mathcal{D}}(f)$, *contains all problems X such that:*

1. *there exists a* uniform *family of P systems $\Pi_X = (\Pi_X(1); \Pi_X(2); \ldots)$ of type $\mathcal{D}$: each $\Pi_X(n)$ can be constructed by a deterministic Turing machine with input n in a time polynomial to n.*
2. *Each $\Pi_X(n)$ is* sound: *$\Pi_X(n)$ starting with a (properly encoded) input $x \in X$ of size n expels out a distinguished object* yes *if and only if the answer to x is "yes".*
3. *Each $\Pi_X(n)$ is* confluent: *all computations of $\Pi_X(n)$ with the same input x of size n give the same result: "yes" or "no".*
4. *Π_X is f-efficient: $\Pi_X(n)$ always halts in at most $f(n)$ steps.*

Alternatively we can consider *semi-uniform families* of P systems $\Pi_X = (\Pi_X(x_1); \Pi_X(x_2); \ldots)$ whose members $\Pi_X(x_n)$ can be constructed by a deterministic Turing machine with input x_n in a polynomial time w.r.t. $|x_n|$. In this case, for each instance of X we have a special P system which therefore does not need an input. The resulting class of problems is denoted by $\mathbf{MC}^S_{\mathcal{D}}(f)$. Obviously, $\mathbf{MC}_{\mathcal{D}}(f) \subseteq \mathbf{MC}^S_{\mathcal{D}}(f)$ for a given class $\mathcal{D}$ and a constructible function f.

Particularly, we denote by

$$\mathbf{PMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_{\mathcal{D}}(\mathcal{O}(n^k)), \qquad \mathbf{PMC}^S_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{MC}^S_{\mathcal{D}}(\mathcal{O}(n^k)),$$

the classes of problems solvable by uniform (semi-uniform, respectively) families of P systems in polynomial time. Let us denote by $\mathcal{MP}$ the class of P systems with proteins on membranes. The following relation follows by [8] for P systems with proteins on membranes:

$$\mathbf{NP} \subseteq \mathbf{PMC}^S_{\mathcal{MP}}. \tag{2}$$

3 Solving QSAT in linear time

In this section we show that P systems with proteins on membranes can solve in linear time the **PSPACE**-complete problem QSAT. More precisely, there exists a semi-uniform family of these P systems such that for each instance of QSAT, a proper P system solving that instance in a linear time can be constructed in a polynomial time w.r.t. the size of the instance. We also observe interesting trade-off between the use of certain elementary P systems operations.

The problem QSAT (satisfiability of quantified propositional formulas) is a standard **PSPACE**-complete problem. It asks whether or not a given quantified boolean formula in the conjunctive normal form assumes the value *true*. A formula as above is of the form

$$\gamma = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m), \tag{3}$$

where each Q_i, $1 \leq i \leq n$, is either $\forall$ or $\exists$, and each C_j, $1 \leq j \leq m$, is a *clause* of the form of a disjunction

$$C_j = y_1 \vee y_2 \vee \ldots \vee y_r,$$

with each y_k being either a propositional variable, x_s, or its negation, $\neg x_s$. For example, let us consider the propositional formula

$$\beta = Q_1 x_1 Q_2 x_2 [(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$$

It is easy to see that it is *true* when $Q_1 = \forall$ and $Q_2 = \exists$, but it is *false* when $Q_1 = \exists$ and $Q_2 = \forall$.

The proof given below is based on the technique already employed in [10] which deals with P systems with active membranes. However, since the function of membrane proteins is different, the proof was substantially adapted. Notice, e.g., that in the P systems with active membranes, the division operation is driven by both membrane contents and polarization, while here it is controlled solely by membrane proteins. As a result, in [10] the membrane structure divides in the bottom-up manner, here the reverse top-down order must be employed.

Theorem 1. $\mathbf{PSPACE} \subseteq \mathbf{PMC}^S_{\mathcal{MP}}$.

Proof. Consider a propositional formula γ of the form (3) with

$$C_i = y_{i,1} \vee \ldots \vee y_{i,p_i},$$

for some $p_i \geq 1$, and $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$, for each $1 \leq i \leq m, 1 \leq j \leq p_i$. We construct the P system

$$\Pi = (O, P, \mu, w_0/z_0, w_1/z_1, \ldots, w_{n+2}/z_{n+2}, \emptyset, R_0, R_1, \ldots, R_{n+2}, 0)$$

with the components

$$\begin{aligned}
O &= \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{r_i, \overline{r}_i \mid 1 \leq i \leq m\} \cup \{t, s\},\\
P &= \{p_0, p_+, p_-, p_x\},\\
\mu &= [_0[_1 \cdots [_n[_{n+1}]_{n+1}[_{n+2}]_{n+2}]_n \cdots]_1]_0,\\
w_0 &= w_{n+2} = \lambda,\\
w_i &= a_i, \text{ for each } i = 1, 2, \ldots, n,\\
w_{n+1} &= r_1 r_2 \ldots r_m,\\
z_0 &= p_0, \quad z_1 = p_x,\\
z_i &= p_0, \text{ for all } i = 2, \ldots, n+2.
\end{aligned}$$

The rules contained in the sets R_i are defined below:

In R_i, $1 \leq i \leq n$:

$$[_i p_x|\,]_i \to [_i p_+|\,]_i [_i p_-|\,]_i, \quad [_i p_+|a_i]_i \to [_i p_+|t_i]_i, \quad [_i p_-|a_i]_i \to [_i p_-|f_i]_i \tag{4}$$

In R_i, $1 \leq i \leq n-1$:

$$t_i[_{i+1} p_0|\,]_{i+1} \to [_{i+1} p_x|t_i]_{i+1}, \quad f_i[_{i+1} p_0|\,]_{i+1} \to [_{i+1} p_x|f_i]_{i+1} \tag{5}$$

In R_i, $3 \leq i \leq n$:

$$t_j[_i p_0|\,]_i \to [_i p_0|t_j]_i, \quad f_j[_i p_0|\,]_i \to [_i p_0|f_j]_i \text{ for all } j,\ 1 \leq j \leq i-2 \tag{6}$$

In R_{n+1} :

$$\begin{aligned}
&t_i[_{n+1} p_0|r_j]_{n+1} \to r_j[_{n+1} p_0|t_i]_{n+1}\\
&\text{for all } i, j,\ 1 \leq i \leq n,\ \leq j \leq m \text{ such that the clause } C_j \text{ contains } x_i
\end{aligned} \tag{7}$$

In R_{n+1} :

$$\begin{aligned}
&f_i[_{n+1} p_0|r_j]_{n+1} \to r_j[_{n+1} p_0|f_i]_{n+1}\\
&\text{for all } i, j,\ 1 \leq i \leq n,\ 1 \leq j \leq m \text{ such that the clause } C_j \text{ contains } \neg x_i
\end{aligned} \tag{8}$$

In R_{n+1} :

$$\begin{aligned}
&[_{n+1} p_0|t_i]_{n+1} \to t_i[_{n+1} p_0|\,]_{n+1}, \quad [_{n+1} p_0|f_i]_{n+1} \to f_i[_{n+1} p_0|\,]_{n+1}\\
&\text{for all } i,\ 1 \leq i \leq n
\end{aligned} \tag{9}$$

In R_{n+2} :

$$r_1[_{n+2} p_0|\,]_{n+2} \to [_{n+2} p_0|\overline{r}_1]_{n+2} \tag{10}$$

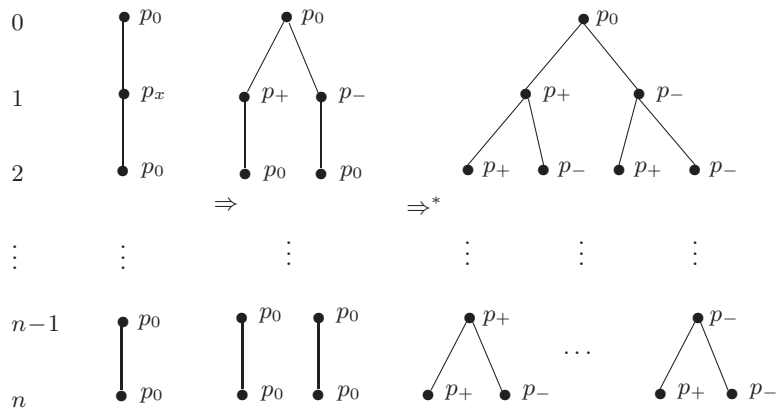

Fig. 2. Expansion of the initial membrane structure into a binary tree (only the first $n+1$ levels shown). The symbols at nodes indicate the proteins present on membranes.

In R_{n+2} :

$$r_{i+1}[_{n+2}p_0|\overline{r}_i]_{n+2} \to \overline{r}_i[_{n+2}p_0|\overline{r}_{i+1}]_{n+2} \text{ for all } i,\ 1 \le i \le n-1 \tag{11}$$

In R_{n+2} :

$$[_{n+2}p_0|\overline{r}_m]_{n+2} \to t[_{n+2}p_0|\ ]_{n+2} \tag{12}$$

In $R_i,\ 1 \le i \le n$ such that $Q_i = \forall$:

$$[_i p_-|t]_i \to s[_i p_-|\ ]_i, \quad s[_i p_+|t]_i \to t[_i p_+|s]_i \tag{13}$$

In $R_i,\ 1 \le i \le n$ such that $Q_i = \exists$:

$$[_i p_-|t]_i \to t[_i p_-|\ ]_i, \quad [_i p_+|t]_i \to t[_i p_+|\ ]_i \tag{14}$$

It is easy to check that the size of the P system Π (the number of objects, membranes, rules, the size of the initial configuration etc.) is $\mathcal{O}(nm)$, n being the number of variables and m the number of clauses. Also the system can obviously be constructed in a polynomial (linear) time.

Initial phase of computation of the system Π is illustrated in Fig. 2. In the first step the non-elementary membrane at level 1 is divided by the first rule in (4) into two parts with different membrane proteins. In the next step, symbols f_1 and t_1 are produced in the two resulting membranes, see the next rules in (4). In the third step, these symbols are moved one level lower, into the membranes labeled 2, see (5). The membrane protein on these membranes is changed to p_x. This cycle is repeated n times and waves corresponding to the division by rules (4) descend the membrane tree towards its leaves. Simultaneously, the produced symbols t_i and f_i move towards the leaves of the tree thanks to the rules (6). This phase is finished after $3n-1$ steps when the membrane structure forms a balanced binary tree, see Fig. 2. Each of its 2^n nodes at level n contains a set

of objects $\{x_1, x_2, \ldots, x_n\}$, where $x_i \in \{f_i, t_i\}$, $1 \leq i \leq n$, such that all possible n-tuples are present.

Second phase consists of checking whether the formula without quantifiers is satisfied by the n-tuples of logical values $(x_1, x_2, \ldots, x_n)$. The checking is done for all the n-tuples in parallel. It starts by moving of those objects r_i, $1 \leq i \leq m$, corresponding to the clauses C_i which are satisfied by a particular n-tuple, from the membrane $[\,]_{n+1}$ to $[\,]_n$. Rules (7)–(9) are responsible for this process. Whenever objects $r_1, \ldots, r_m$ appear in membrane $[\,]_n$, another process starts whose purpose is to check whether all r_i, $1 \leq i \leq m$, are present. This is done by their movements to-and-from membrane $[\,]_{n+2}$ driven by rules (10)–(12). Eventually, object t is released into the membrane $[\,]_n$.

The application of rules of the second phase can partially overlap with the initial phase: whenever first objects t_i or f_i arrive into the membrane $[\,]_n$, the second phase starts, while remaining t_i's and f_i's can arrive later. However, the application of the rules in the second phase described above is not altered.

Finally, third phase of computation checks whether the whole formula with quantifiers is satisfied. Objects t move upwards the membrane structure tree, checking at each level one quantifier $\forall$ or $\exists$. Observe that rules (13)–(14) allow for existence of more than one symbol t per membrane (in the case of $\exists$) which, however, do not alter the computation. Eventually, object t appears in membrane 0, signaling that the formula is satisfied, and the system halts.

The whole computation is performed by time linearly limited from above by the values of n and m. More specifically, the initial phase is finished in $3n - 1$ steps, the second phase takes up to $3m$ steps and the last phase up to $2n$ steps. In total, the computation takes $\mathcal{O}(n + m)$ steps.

Observe that rules (5) are the only rules of type $2cp$. All the rest are restricted (or division) rules. Furthermore, these $2cp$ rules are used only to control the membrane division process. The membrane division rules can be controlled solely by the presence of a specific membrane protein. Assume that we introduced division rules similar as in P systems with active membranes, i.e., of type $[\,_i p|a]\,_i \rightarrow [\,_i p|b]\,_i [\,_i p|c]\,_i$, controlled by the presence of certain object in a membrane. Then the rules (5) would not be needed and the whole P systems could use only restricted and division rules.

Hence, it turns out that the only necessary purpose of membrane proteins is the control of membrane division forced by the specific type of division rules. If we compare our proof with that in [10], we observe that the role played in [10] by the membrane polarization (which is in some sense generalized in the concept of membrane proteins) is in our proof frequently replaced by the use of antiport rules of types (4) and (5). Therefore, there is a trade-off between membrane labeling (polarization, proteins) and antiport rules.

This suggests that from the point of view of efficiency, there is no substantial difference between restricted and "change protein" rules. The paper [8] shows that the universality can be reached only with the restricted rules, too. However, there is another trade off between the number of membranes and the use of "change protein" rules in this case.

4 Simulation of a P system with proteins on membranes in polynomial space

In this section we demonstrate an algorithm for simulation of P systems with proteins on membranes which proves the relation reverse to that given in Theorem 1. Notice that the simulated P system is confluent (hence possibly nondeterministic), therefore the conditions of the Parallel Computation Thesis are satisfied. However, our simulation itself is deterministic – at each step we simulate only one chosen multiset of applicable rules. Hence we simulate one possible sequence of configurations of the P system. The algorithm of selection of the rules to be applied corresponds to introducing a weak priority between rules: (i) bottom-up priority between rules associated to different membranes, (ii) priority between rules in the same membrane, given by the order in which they are listed, including the priority between types 1–6, in this order. The confluency condition ensures that such a simulation leads always to a correct result.

We employ the technique of reverse-time simulation which is known from the general complexity theory when dealing with the second class machines. Instead of simulating a computation of a P system from its initial configuration onwards (which could require an exponential space for storing configurations), we create the recursive function `State` which returns the state of any membrane h after a given number of steps. The recursive calls evaluate contents of the membranes interacting with h in a reverse time order (towards the initial configuration). The key observation is that the state of the membrane is determined by its own state, states of the embedded membranes and its parent membrane at the previous computational step. In such a manner we do not need to store a state of any membrane, but instead we calculate it recursively whenever it is needed. The depth of the recursive calls is proportional to the number of steps of the simulated P system. Furthermore, at each level of the call stack we must store a state of a single membrane which can be done in a polynomial space. In this way a result of any $T(n)$-time-bounded computation of a confluent accepting P system with proteins on membranes can be found in a space polynomial to $T(n)$.

Theorem 2. $\mathbf{PMC}^S_{\mathcal{MP}} \subseteq \mathbf{PSPACE}$.

Proof. Let $\Pi = (O, P, \mu, w_1/z_1, \ldots, w_n/z_n, E, R_1, \ldots, R_n, i_0)$ be a membrane system. We define *state* of each membrane h of μ as a pair $S = (M, Z)$, where M is the multiset characterizing the contents of (the region enclosed by) membrane h and Z is the multiset of proteins on membrane h. We use the notation $S.M$ and $S.Z$ to refer to these two components of state.

The state of a membrane h at certain computational step can be computed recursively from the state of h, the state of its parent membrane and states of all its embedded membranes at the previous step. We take into the account all membranes which could be possibly created by maximal membrane division within the system. The algorithm can be described informally as follows:

– verify whether our membrane h exists at all at given computational step; this is done by checking

  - the existence of membrane h in the previous step;
  - a possible application of a division rule in membrane h in the previous step;
  - the existence of a parent membrane of h;
- calculate recursively the state of membrane h at the end of the previous step;
- calculate recursively previous states of all the membranes directly embedded in h, then enumerate objects which h sends to/ receives from these membranes during the given step;
- calculate recursively previous state of the parent membrane (containing h), unless h is the skin membrane;
- simulate an application of rules of types 1–6 in membrane h during the given step.

Special attention must be paid to the skin membrane. Recall that some objects (not present in E) can be expelled into the outer region and later eventually return back to the system. Therefore, we must keep track of the contents of the outer region, too. Therefore, in the algorithm we assume the existence of an "outer membrane" at level 0, increasing the depth of the membrane structure by 1. This virtual membrane is the parent of the skin membrane, with no rules and no proteins assigned to it. Initially it contains infinitely many copies of objects from E, later there might appear objects from $O \setminus E$ which were expelled from the skin membrane.

We assume without loss of generality that the original labeling of membranes of Π in μ is one-to-one. However, during the computation of Π the membranes may be divided, keeping their original labels. Hence there may exist more membranes with the same label. To identify membranes uniquely, we add to each label an index in square brackets. Observe that in one computational step a division may simultaneously take place at various levels (possibly all) of the membrane structure tree. Therefore, a membrane structure of a depth n can potentially produce $\mathcal{O}(2^n)$ membranes in a single step, and we must be able to identify all of them. The indices are assigned due to the following rules:

1. The skin membrane has always an empty index.
2. The index of each membrane at a nesting level $k+1$ after n steps of computation consists of k n-tuples of numbers 1 or 2, for $k, n \geq 0$. In the initial configuration each index is empty.
3. After each computational step, indices are extended in a top-down manner. Consider a membrane $h[i_{11} \dots i_{1(n-1)}, \dots, i_{k1} \dots i_{k(n-1)}]$. If h does not divide at step n, digit 1 is attached to the last $(n-1)$-tuple. If h is divided, the resulting two membranes have attached 1 and 2, respectively, to their last $(n-1)$-tuples.
4. Simultaneously the same digit is attached to the k-th tuple of indices of all sub-membranes of h.

Instead of the detailed notation of indices $i_{11} \dots i_{1n}, \dots, i_{k1} \dots i_{kn}$ we will in the sequel use also its shorter version $i_{11} \dots i_{kn}$. The membrane indexing is illustrated in Fig. 3. At the first step, membrane d was divided. At the second step, the non-elementary membrane $c_{1,1}$ was divided. Observe the following facts:

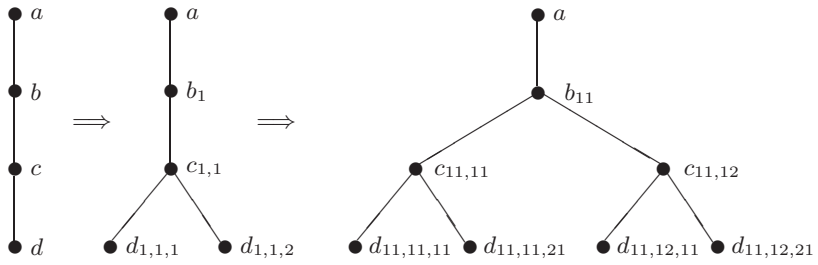

Fig. 3. Example of an indexed membrane structure after two computational steps

- An index of a membrane contains as prefixes the indices of all its parent membranes, up to the skin membrane.
- The parent membrane of $h[i_{11} \dots i_{kn}]$ has always the index $i_{11} \dots i_{(k-1)n}$.
- A membrane $h[i_{11} \dots i_{kn}]$ evolved from membrane $h[i_{11} \dots i_{k(n-1)}]$ at n-th step of computation.
- Given an initial membrane structure μ and a number $n \geq 0$, we can effectively enumerate all the membranes which could potentially exist in μ after n steps. Certain indices can denote membranes non-existing due to non-application of membrane division rules.

Now we describe the function `State` which computes the state of any membrane h of Π at a given step of computation.

Function `State`

Parameters: $h[i_{11} \dots i_{kn}]$

Local variables:
S – state of the membrane $h[i_{11} \dots i_{k(n-1)}]$,
S' – state of the membrane $h[i_{11} \dots i_{kn})]$,
T – state of the parent membrane of $h[i_{11} \dots i_{k(n-1)}]$,
T' – state of the parent membrane of $h[i_{11} \dots i_{kn}]$,

1. If $n = 0$ then return the state of membrane h in the initial configuration and exit.
2. If $k > 0$ (i.e. h is not the virtual "outer membrane"), then
   / We check the existence of membrane $h[i_{11} \dots i_{kn}]$ */*
   (a) $T \leftarrow$ `State`(`Parent`($h[i_{11} \dots i_{kn}]$))
   / We calculate the state of the parent membrane of $h[i_{11} \dots i_{kn}]$ */*
   (b) If $T = nil$ then return nil and exit.
   / If the parent membrane does not exist, neither does its child $h[i_{11} \dots i_{kn}]$ exist. */*

(c) $S \leftarrow \texttt{State}(h[i_{11} \dots i_{k(n-1)}])$

(d) If $S = nil$ then return nil and exit.
/ If membrane $h[i_{11} \dots i_{k(n-1)}]$ did not exist after $(n-1)$ steps, then after n steps its successor $h[i_{11} \dots i_{kn}]$ cannot exist. */*

3. $T' \leftarrow (\emptyset, \emptyset)$, $S' \leftarrow (\emptyset, \emptyset)$
4. $\texttt{Contribution\_from\_children}(h[i_{11} \dots i_{kn}], S, S')$
5. If $k = 0$ then $T \leftarrow (\emptyset, \emptyset)$ else $T \leftarrow \texttt{State}(\texttt{Parent}(h[i_{11} \dots i_{k(n-1)}]))$.
/ We calculate a state of the parent membrane at $(n-1)$-th step. */*
6. */* Now we simulate the evolution of membrane $h[i_{11} \dots i_{k(n-1)}]$ at n-th step. */*
   (a) $\texttt{Try\_rules\_1}(h, S, S', T, T')$
   (b) $\texttt{Try\_rules\_2}(h, S, S', T, T')$
   (c) $\texttt{Try\_rules\_3}(h, S, S', T, T')$
   (d) $\texttt{Try\_rules\_4}(h, S, S', T, T')$
   (e) $\texttt{Try\_rules\_5}(h, S, S', T, T')$
   (f) $\texttt{Try\_rules\_6}(h, S, S', i_{kn})$
7. If $i_{kn} = 2$ and a rule of type 6 was not applied, then $S' \leftarrow nil$.
/ If $i_{nk} = 2$, then membrane $h[i_{11} \dots i_{kn}]$ could only be created by an application of a rule of type 6 during the n-th step. If such a rule was not applied, then $h[i_{11} \dots i_{kn}]$ does not exist. */*
8. If $S' \neq nil$ then $S'.M \leftarrow S'.M \cup S.M$, $S'.Z \leftarrow S'.Z \cup S.Z$.
/ All unused objects and proteins pass unchanged to the next step. */*
9. Return S' and exit.

The following simple function just returns the parent membrane of the membrane specified by a parameter, enriched with indices denoting the same computational step n an the depth $k-1$ of the parent membrane.

Function `Parent`
Parameters:
$h[i_{11} \dots i_{kn}]$ – a membrane whose parent is searched for

1. Return $g[i_{11} \dots i_{(k-1)n}]$, where g is the parent membrane of h in the initial membrane structure μ.

The following procedure calculates the contribution of the children membranes (i.e., the objects they expel) embedded in a membrane h during n-th computational steps. The procedure is recursive since during the n-th step, the rules are applied in the membrane structure tree in the bottom-up priority. Hence

the rules moving objects through elementary membranes may influence objects in the embedding membranes, which in turn influence communication rules in membranes one level above, et cetera. Observe that, with the aid of function `State`, we can uniquely determine the children (in terms of the membrane structure tree) of a given membrane $h[i_{11} \dots i_{kn}]$ existing after $n-1$ computational steps without storing the whole membrane structure of Π after $(n-1)$-th step.
Procedure `Contribution_from_children`

/ Calculates the interaction of a membrane $h[i_{11} \dots i_{kn}]$ with its children membranes at step n by:*
– sending and receiving objects into/from children membranes by rules of type 2–5,
*– modifying objects by proteins on children membranes by rules of type 1. */*
Parameters:
$h[i_{11} \dots i_{kn}]$ – a membrane to which its children contribute
S – an initial state of the parent membrane
S' – a final state of the parent membrane

Local variables: T, T'

For each children membrane g of h in the initial membrane structure μ, and for each n-tuple $j_{(k+1)1} \dots j_{(k+1)n}$, $j_{(k+1)\ell} \in \{1,2\}$, $1 \leq \ell \leq n$:

1. $T \leftarrow \texttt{State}(g[i_{11} \dots i_{k(n-1)}, j_{(k+1)1} \dots j_{(k+1)(n-1)}])$, $\quad T' \leftarrow (\emptyset, 0)$
2. If $T = nil$ then skip the remaining steps.
3. `Contribution_from_children`$(g[i_{11} \dots i_{kn}, j_{(k+1)1} \dots j_{(k+1)n}], T, T')$
   / Calculate recursively the contribution of children membranes to membrane $g[i_{11} \dots i_{kn}, j_{(k+1)1} \dots j_{(k+1)n}]$: we have chosen to evaluate the influence of children membranes prior to the activity of the parent membrane and we must keep this principle on all levels of the membrane tree. */*
4. `Try_rules_1`(g, T, T', S, S')
5. `Try_rules_2`(g, T, T', S, S')
6. `Try_rules_3`(g, T, T', S, S')
7. `Try_rules_4`(g, T, T', S, S')
8. `Try_rules_5`(g, T, T', S, S')

Observe that the parent of $h[i_{11} \dots i_{kn}]$ is the membrane $g[i_{11} \dots i_{(k-1)n}]$, where g is the parent of h in the initial state. Then one can deduce that the recursive function `State` is defined correctly because each recursive call during the computation of $\texttt{State}(h[i_{11} \dots i_{kn}])$ is in one of the forms

$$\texttt{State}(g[i_{11} \dots i_{(k-1)n}]), \tag{15}$$

$$\texttt{State}(g[i_{11} \dots i_{(k-1)(n-1)}]), \tag{16}$$
$$\texttt{State}(h[i_{11} \dots i_{k(n-1)}]), \tag{17}$$

where d is the depth of the initial membrane structure μ. By (15)–(17), each such call decreases the value of at least one of the parameters $n, k \geq 0$.

Furthermore, in the procedure `Contribution_from_children` one needs to evaluate $\texttt{State}(g[i_{11} \dots i_{k(n-1)}, j_{(k+1)1} \dots j_{(k+1)(n-1)}])$. However, since k is bounded from above, the resulting graph of recursive calls is acyclic and finite. See also equations (19)–(23) which describe mathematically the structure of calls.

Finally, the following procedures evaluate a simultaneous application of a specific type of rules in a membrane specified by a parameter. The rules can move objects through the membrane and hence can influence also the state of the parent membrane.

Procedures `Try_rules_1` – `Try_rules_6`
/ Apply a given type of rules attached to a membrane h; remove their left-hand sides from initial state(s) S (T) and add their right-hand sides to final state(s) S' (T').*

We treat restricted rules as a special case of the generalized ones for $p = p'$./*

Parameters:
h – label of the membrane processed
S – initial state of the membrane
S'– final state of the membrane
T – initial state of the parent membrane (only rules 1–5)
T' – final state of the parent membrane (only rules 1–5)
i – index of a divided membrane, value 1 or 2 (only rules 6)

(1a) For each rule $[_h p|a]_h \to [_h p'|b]_h$ in R_h :
- let $j = \min\{|S.M|_a, |S.Z|_p)\}$
- remove j occurrences of a from $S.M$
- remove j occurrences of p from $S.Z$
- add j occurrences of b to $S'.M$
- add j occurrences of p' to $S'.Z$

(1b) For each rule $a[_h p|]_h \to b[_h p'|]_h$ in R_h :
- let $j = \min\{|T.M|_a, |S.Z|_p)\}$
- remove j occurrences of a from $T.M$
- remove j occurrences of p from $S.Z$
- add j occurrences of b to $T'.M$
- add j occurrences of p' to $S'.Z$

(2) See (3a), (3b) since rules of type (2) can be treated as a special case of (3).

(3a) For each rule $[_h p|a]_h \to b[_h p'|]_h$ in R_h :
– let $j = \min\{|S.M|_a, |S.Z|_p)\}$
– remove j occurrences of a from $S.M$
– remove j occurrences of p from $S.Z$
– add j occurrences of b to $T'.M$
– add j occurrences of p' to $S'.Z$

(3b) For each rule $a[_h p|]_h \to [_h p'|b]_h$ in R_h :
– let $j = \min\{|T.M|_a, |S.Z|_p)\}$
– remove j occurrences of a from $T.M$
– remove j occurrences of p from $S.Z$
– add j occurrences of b to $S'.M$
– add j occurrences of p' to $S'.Z$

(4) See (5) since rules of type (4) can be treated as a special case of (5).

(5) For each rule $a[_h p|b]_h \to c[_h p'|d]_h$ in R_h :
– let $j = \min\{|T.M|_a, |S.M|_b, |S.Z|_p)\}$
– remove j occurrences of a from $T.M$
– remove j occurrences of b from $S.M$
– remove j occurrences of p from $S.Z$
– add j occurrences of c to $T'.M$
– add j occurrences of d to $S'.M$
– add j occurrences of p' to $S'.Z$

(6) For each rule $[_h p|]_h \to [_h p'|]_h[_h p''|]_h$ in R_h :
if $p \in S.Z$ then begin
– remove p from $S.Z$
– if $i = 1$ then add p' to $S.Z'$ else add p'' to $S.Z'$
– skip all other rules of type (6)
end
/ Observe that the contents of the membrane S is unchanged during the division as S' represents only one of the two resulting copies */*

Complexity of the simulation procedure

We specify the consumption of resources (namely the computational space) required to solve an instance of a size s of a decision problem. Let the instance be solved by a confluent P system $\Pi = (O, P, \mu, w_1/z_1, \ldots, w_m/z_m, E, R_1, \ldots, R_m, i_0)$ of a size $s^{\mathcal{O}(1)}$, a member of a semi-uniform family. A result of its computation can be calculated with the aid of the function `State`. One can subsequently calculate $\texttt{State}(i_0, n)$, until the object *yes* appears in this membrane or until the computation halts. Halting can be tested by computing $\texttt{State}(h, n)$ for all the membranes h which could potentially exist after n steps, $n = 0, 1, 2, \ldots$, until no rule can be applied in any of them. We determine the space complexity of the function `State`. Let

d be the depth of the initial membrane structure tree μ,

$q = \mathrm{card}(O)$,
o_n denote the number of objects within the system after n steps. Hence, $o_0 = |w_1| + \ldots + |w_m|$.

By the assumption, the values of m, d, q and $\log o_0$ are bounded from above by $s^{\mathcal{O}(1)}$ (which is the initial size of Π). In the rest of the proof we treat them as constants as they are fixed for a given Π. Evaluating o_n, one observes that the number of objects can increase only during the membrane can division, and their number after n steps is bounded by the expression $m(2^d)^n$ (if each membrane except the skin is at every step divided). Hence the total number of the objects in the system is

$$o_n \leq o_0 m (2^d)^n.$$

The maximal number of objects in a single membrane grows exponentially, too, as all can eventually concentrate in the same membrane. The space (in bits) necessary to store the contents of an arbitrary membrane after n steps is

$$b_n \leq q\lceil \log o_n \rceil \leq q\lceil \log(o_0 m) \rceil + qdn = c_0 + c_1 n \tag{18}$$

for positive constants c_0 and c_1 of size $s^{\mathcal{O}(1)}$.

Functions `State`, `Parent` and `Contribution_from_children` with a parameter $h[i_{11} \ldots i_{kn}]$ store the following information:

1. a specification of membrane $h[i_{11} \ldots i_{kn}]$ which requires $kn + \lceil \log m \rceil$ bits,
2. variables as S, S', T, T' which store the content of membrane $h[i_{11} \ldots i_{kn}]$ (or its parent/child) and each of which requires at most b_n bits.

We do not need to consider parameters passed to procedures `Try_rules_X` as all the structured parameters S, S', T, T' are passed by reference.

Denote the space complexity of the functions `Contribution_from_children` and `State` with the parameter $h[i_{11} \ldots i_{kn}]$ by $C(n,k)$ and $S(n,k)$, respectively. Observe that this complexity does not depend on a particular membrane h but solely on the values of n and k. The structure of mutual calls of these procedures and the variables they use correspond to the following recurrences:

$$S(0,k) = b_0, \quad 0 \leq k \leq d \tag{19}$$

$$S(n,0) = \max\{C(n,0), S(n-1,0)\} + 4b_n + c, \quad n \geq 1 \tag{20}$$

$$S(n,k) = \max\{C(n,k), S(n-1,k), S(n-1,k-1)\} + 4b_n + kn + c, \quad n \geq 1,\ 1 \leq k \leq d \tag{21}$$

$$C(n,d) = 0 \tag{22}$$

$$C(n,k) \leq \max\{C(n,k+1), S(n-1,k+1)\} + 4b_n + n(k+1) + c, \quad 0 \leq k < d, n \geq 0 \tag{23}$$

By expanding (23) to a series for $k, k+1, \ldots, d$ we obtain

$$C(n,k) \leq \max\{S(n-1,i) \mid k < i \leq d\} + \mathcal{O}(d^2 n + d b_n) \tag{24}$$

for $0 \leq k \leq d$, $n \geq 0$. Let us define

$$S(n) = \max\{S(n,k) \mid 0 \leq k \leq d\}. \tag{25}$$

By (24) and (25) we can rewrite (21) in the form

$$\begin{aligned} S(n,k) \leq \max\{S(n-1) + \mathcal{O}(d^2 n + db_n)\} + \mathcal{O}(b_n + kn), \\ n \geq 1,\ 1 \leq k \leq d \end{aligned} \tag{26}$$

By substituting $S(n,i)$, $i = k-1, k-2, \ldots, 0$ with (26), we can expand (26) as follows:

$$\begin{aligned} S(n,k) &\leq \max\{S(n,k-1), S(n-1) + \mathcal{O}(d^2 n + db_n)\} + 2\mathcal{O}(b_n + kn) \\ &\leq \max\{S(n,k-2), S(n-1) + \mathcal{O}(d^2 n + db_n)\} + 3\mathcal{O}(b_n + kn) \\ &\vdots \\ &\leq \max\{S(n,0), S(n-1) + \mathcal{O}(d^2 n + db_n)\} + (k+1)\mathcal{O}(b_n + kn) \\ &\leq \max\{C(n,0) + 4b_n + c, S(n-1) + \mathcal{O}(d^2 n + db_n)\} + \mathcal{O}(kb_n + k^2 n) \\ &\leq S(n-1) + \mathcal{O}(d^2 n + db_n). \end{aligned}$$

The next-to-last step was done by substituting $S(n,0)$ with (20). In the last step we substituted $C(n,0)$ with (24) and (25). Therefore, the recurrence (19)–(21) can be rewritten with the aid of (25) as follows:

$$\begin{aligned} S(0) &= b_0 \\ S(n) &\leq S(n-1) + \mathcal{O}(d^2 n + db_n) \end{aligned}$$

A solution to this recurrence is $S(n) = \mathcal{O}(d^2 n^2 + ndb_n)$. Recall that $d = s^{\mathcal{O}(1)}$, where s is the original instance size. By (18) we get

$$S(n) = (sn)^{\mathcal{O}(1)}. \tag{27}$$

If we assume that Π is polynomial time-bounded, we obtain also $n = s^{\mathcal{O}(1)}$. After substituting to (27) one can conclude that the simulation is done in **PSPACE**. □

If we put together Theorems 2 and 1, we obtain the parallel computation thesis for semi-uniform families of confluent P systems with proteins on membranes:

Corollary 1. $\mathbf{PMC}^S_{\mathcal{MP}} = \mathbf{PSPACE}$.

5 Discussion

We have shown that semi-uniform families of P systems with proteins on membranes can solve in polynomial time exactly the class of problems **PSPACE**. Therefore, they are computationally equivalent to other parallel computing model as PRAM or alternating Turing machine. We conjecture that the same result holds with regards to *uniform* families of P systems but no formal proof is known

yet. Possibly a construction similar to that in [2] could be used to solve this problem. Also the characterization of power of *non-confluent* P systems with proteins membranes remains open. The presented proof cannot be simply adapted to this case by using a non-deterministic Turing machine. The reason is that we cannot store non-deterministic choices of such a P system along a chosen trace of computation, as this would require an exponential space. Therefore, we do not know what is the power of non-confluent P systems with proteins on membranes.

A similar result has been previously shown in [11] for the case of P systems with active membranes. Therefore, taking into the account another results of this kind related to other types of natural or molecular computing, one could suggest that the class **PSPACE** represents natural characterization of deterministic natural computations. It is important to note that certain operations used in P systems with proteins on membranes, as the division of non-elementary membranes, seem to have in practice very limited scalability, on one hand. On the other hand, certain properties of biocomputing models, as the massive parallelism, minimal energy consumption, microscopic dimensions of computing elements etc. makes it very attractive to seek for ways how to harness the micro-biological machinery for algorithmic tasks.

Among further problems we mention restricted variants of the P systems with proteins on membranes. How would the computational power of (semi)uniform families of such systems change if only certain types of rules were allowed?

Acknowledgements

Research was partially supported by the National Science Foundation Grant CCF-0523572, INBRE Program of the NCRR (a division of NIH), support from CNCSIS grant RP-13, support from CNMP grant 11-56 /2007, support from the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2006-15595 and the program I3, and by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group).

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, New York, 4th edition, 2002.
2. A. Alhazov, C. Martín-Vide, and L. Pan. Solving a PSPACE-complete problem by P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
3. L. Cardelli. Brane calculi – interactions of biological membranes. In *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–280, Berlin, 2005. Springer-Verlag.
4. P. Frisco. *Computing with Cells. Advances in Membrane Computing*. Oxford University Press, Oxford, 2009.
5. Andrei Paun and Gheorghe Paun. The power of communication: P systems with symport/antiport. *New Generation Comput.*, 20(3):295–306, 2002.

6. M.J. Pérez-Jiménez, A.R. Jiménez, and F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2:265–285, 2003.
7. A. Păun and B. Popa. P systems with proteins on membranes. *Fundamenta Informaticae*, 72(4):467–483, 2006.
8. A. Păun and B. Popa. P systems with proteins on membranes and membrane division. In O.H. Ibarra and Z. Dang, editors, *DLT 2006*, volume 4036 of *Lecture Notes in Computer Science*, pages 292–303, Berlin, 2006. Springer-Verlag.
9. G. Păun. *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
10. P. Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, 2003.
11. P. Sosík and A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *J. Comput. System Sci.*, 73(1):137–152, 2007.
12. P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier, Amsterdam, 1990.
13. The P systems web page. `http://ppage.psystems.eu/`.