P Systems with Active Membranes Characterize PSPACE

Petr Sosík [a,b,*] Alfonso Rodrígues-Patón Aradas [a]

[a] *Universidad Politécnica de Madrid – UPM, Faculdad de Informática*
Campus de Montegancedo s/n, Boadilla del Monte
28660 Madrid, Spain

[b] *Institute of Computer Science, Silesian University*
74601 Opava, Czech Republic

Abstract

P system is a natural computing model inspired by behavior of living cells and their membranes. We show that (semi-)uniform families of P systems with active membranes can solve in polynomial time exactly the class of problems **PSPACE**. Consequently, these P systems are computationally equivalent (w.r.t. the polynomial time reduction) to standard parallel machine models as PRAM and the alternating Turing machine.

Key words: Keywords: Natural computing, P system, PSPACE, alternating Turing machine

1 Introduction

The computational power of P systems in active membranes was studied first in [9], where their ability to solve NP-complete problems in polynomial time was demonstrated. Later it was shown in [12,2] that (semi-)uniform families of deterministic P systems with active membranes can solve also the problem QBF (satisfiability of quantified propositional formulas) in a polynomial time. As QBF is a classical **PSPACE**-complete problem, any other problem in PSPACE can be also (after a reduction to QBF) solved in a polynomial time in the same way.

* Corresponding author.
Email addresses: petr.sosik@fpf.slu.cz (Petr Sosík), arpaton@fi.upm.es (Alfonso Rodrígues-Patón Aradas).

Here we complete the characterization of the computational power of P systems with active membranes. We show that the class of problems solvable in polynomial time by these P systems is exactly the class PSPACE. As a consequence, these P systems satisfy the so-called *Parallel Computation Thesis* [6] for a computer M :

$$M\text{-TIME}(T^{\mathcal{O}(1)}(n)) = \text{SPACE}(T^{\mathcal{O}(1)}(n)). \tag{1}$$

Computers satisfying this thesis form the *second machine class* [13,4]. Among its members there are standard models of parallel computers as SIMDAG (known also as SIMD PRAM), APM – a model of existing vector supercomputers [14], P-RAM – a model of MIMD PRAM computer, alternating Turing machine and others [4]. An interesting member is the genetic Turing machine [11], a computational model of genetic crossing-over. It is also straightforward that the second machine class is closed under the polynomial-time reduction.

To demonstrate that any $f(n)$ time-bounded computation of a P system with active membranes can be simulated in a space polynomial w.r.t. $f(n)$, we adopt the technique of *reverse-time* simulation. Instead of simulating a computation of a P system from its initial configuration onwards (which would require an exponential space for storing configurations), we create a recursive function 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). 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.

2 Definitions

In this section we give a brief description of P systems with active membranes due to [9] or [8], where also more details can be found. A *membrane structure* is represented by a Venn diagram (or a rooted tree) and is identified by a string of correctly matching parentheses, with a unique external pair of parentheses corresponding to the external membrane, called *the skin*. A membrane without any other membrane inside is said to be *elementary*. The following example from [9] illustrates the situation: the membrane structure at Figure 1 is identified by the string

$$\mu = [_1[_2[_5\,]_5[_6\,]_6]_2[_3\,]_3[_4[_7[_8\,]_8]_7]_4]_1.$$

In what follows, we interpret the membrane structure of Π as a rooted tree and refer occasionally to its elements – membranes as nodes in this tree. The

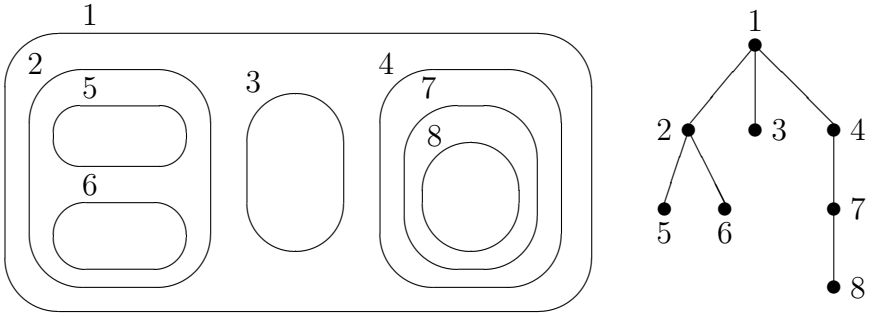

Fig. 1. A membrane structure and its associated tree.

membranes can be further marked with + or −, and this is interpreted as an "electrical charge", or with 0, and this means "neutral charge". We will write $[_i\,]_i^+, [_i\,]_i^-, [_i\,]_i^0$ in the three cases, respectively.

The membranes delimit *regions*, precisely identified by the membranes (the region of a membrane is delimited by the membrane and all membranes placed immediately inside it, if any such a membrane exists). In these regions we place *objects*, which are represented by symbols of an alphabet. Several copies of the same object can be present in a region, so we work with *multisets* of objects. A multiset m over an alphabet V can be represented by any string $x \in V^*$ (by V^* we denote the free monoid generated by V with respect to the concatenation and the identity λ) such that the number of occurrences of a symbol $a \in V$ in x represents the multiplicity of the object a in the multiset m.

A P system with active membranes is a construct

$$\Pi = (V, H, \mu, w_1, \ldots, w_m, R),$$

where:

(i) $m \geq 1$;
(ii) V is an alphabet;
(iii) H is a finite set of *labels* for membranes;
(iv) μ is a *membrane structure*, consisting of m membranes, labelled (not necessarily in a one-to-one manner) with elements of H; all membranes in μ are supposed to be neutral;
(v) $w_1, \ldots, w_m$ are strings over V, describing the *multisets of objects* placed in the m regions of μ;
(vi) R is a finite set of *developmental rules*, of the following forms:
  (a) $[_h a \to v]_h^\alpha$,
  for $h \in H, \alpha \in \{+, -, 0\}, a \in V, v \in V^*$

(object evolution rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part to the application of these rules nor are they modified by them);

(b) $a[_h\,]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$
(communication rules; an object is introduced into the membrane, maybe modified during this process; also the polarization of the membrane can be modified, but not its label);

(c) $[_h a\,]_h^{\alpha_1} \to [_h\,]_h^{\alpha_2} b$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$
(communication rules; an object is sent out of the membrane, maybe modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[_h a\,]_h^{\alpha} \to b$,
for $h \in H, \alpha \in \{+, -, 0\}, a, b \in V$
(dissolving rules; in reaction with an object, a membrane can be dissolved, leaving all its object in the surrounding region, while the object specified in the rule can be modified);

(e) $[_h a\,]_h^{\alpha_1} \to [_h b\,]_h^{\alpha_2} [_h c\,]_h^{\alpha_3}$,
for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in V$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, maybe of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; all the other objects are copied into both resulting membranes);

(f) $[_{h_0}[_{h_1}\,]_{h_1}^{+} \ldots [_{h_k}\,]_{h_k}^{+}\,[_{h_{k+1}}\,]_{h_{k+1}}^{-} \ldots [_{h_n}\,]_{h_n}^{-}]_{h_0}^{\alpha_2}$
$\to [_{h_0}[_{h_1}\,]_{h_1}^{\alpha_3} \ldots [_{h_k}\,]_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5}\,[_{h_0}[_{h_{k+1}}\,]_{h_{k+1}}^{\alpha_4} \ldots [_{h_n}\,]_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$,
for $n > k \geq 1$, $h_i \in H$, $0 \leq i \leq n$, and $\alpha_2, \ldots, \alpha_6 \in \{+, -, 0\}$;
(division of non-elementary membranes; this is possible only if a membrane contains two immediately lower membranes of opposite polarization, + and −; the membranes of opposite polarizations are separated in the two new membranes, but their polarization can change; always, all membranes of opposite polarizations are separated by applying this rule;
if the membrane labelled h_0 contains other membranes than $h_1, \ldots, h_n$ specified above, then they must have neutral charges in order to make this rule applicable; these membranes are duplicated and then become part of the content of both copies of membrane h_0).

All the above rules are applied in parallel, but at one step, an object a can be a subject of only one rule of type (a)–(e) and a membrane h can be subject of only one rule of type (b)–(f). In the case of rules of type (f) this means that none of the membranes labelled $h_0, \ldots, h_n$ listed in the rule can be simultaneously subject of another rule of type (b)–(f). However, if the membrane

labelled h_0 contains other membranes with neutral charge, they can be simultaneously subject of another rules and the results are copied to both copies of membrane h_0. In general, an application of rules is performed as follows:

(1) any object and membrane which can evolve by a rule of any form, should evolve;
(2) all objects and membranes which cannot evolve pass unchanged to the next step;
(3) if a rule of type (d), (e) or (f) is applied to a membrane, then rules of type (a) are applied first to its objects and then the resulting objects are further copied/moved in accordance with the (d), (e) or (f) type rule;
(4) rules of type (d), (e), (f) are applied during one step in a bottom-up manner: first, they are applied to elementary membranes, then to their parent membranes etc., towards the skin membrane;
(5) the skin membrane can neither be dissolved, nor divided, nor it can introduce an object from outside (unless stated otherwise). Therefore, we assume that there are only rules of types (a) and (c) associated with the skin membrane.

The membrane structure of the system at a given moment, together with all multisets of objects associated with the regions of its membrane structure form the *configuration* of the system. The $(m + 1)$-tuple $(\mu, w_1, \ldots, w_m)$ is the *initial configuration*. We can pass from a configuration to another one by using the rules from R according to the principles given above. Notice that the depth of the membrane structure *cannot grow* during any computation. The computation stops when there is no rule which can be applied to objects and membranes in the last configuration. The result of the computation is the collection of objects expelled from the skin membrane during the whole computation.

In the case of P systems solving decision problems, a distinguished membrane contains at the beginning of computation an input – a description of an instance of a problem. Alternatively, the input can be supplied from outside through the skin membrane. The result of computation (a solution to the instance) is yes if a distinguished object *yes* has been expelled during the computation, otherwise the result is no.

3 Complexity classes of P systems

"Classical" machine models run programs with an arbitrary combination of instruction, with variables storing an arbitrary integer etc. However, when dealing with biocomputing models, the "program" and some of its variables are built in the structure of the system which is often not so flexible. Hence

it is more natural to consider *families* of P systems for solving computational problems [8]. In this manner there have been defined complexity classes for various types of P system [10]. For a decision problem A we denote by $A(n)$ its instances of size n.

Definition 1 *Let X be a class of membrane systems and let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a total function. The* class of problems solved by uniform families of X-type P systems in time f, *denoted by* $\mathbf{MC}_X(f)$, *contains all problems A such that:*

(1) there exists a uniform *family of P systems $\Pi_A = (\Pi_A(1); \Pi_A(2); \ldots)$ of type X whose members $\Pi_A(n)$ can be constructed by a Turing machine with the input n in a polynomial time.*
(2) Each $\Pi_A(n)$ is sound: *there exists a distinguished object* yes *such that $\Pi_A(n)$ starting with the input $A(n)$ expels out object* yes *if and only if the answer to $A(n)$ is "yes".*
(3) Each $\Pi_A(n)$ is confluent: *all computations of $\Pi_A(n)$ with the same input $A(n)$ have the same result "yes" or "no".*
(4) Π_A is f-efficient: $\Pi_A(n)$ always halts in at most $f(n)$ steps.

Alternatively we can consider *semi-uniform families* of P systems $\Pi_A = (\Pi_A(A(1)); \Pi_A(A(2)); \ldots)$ whose members $\Pi_A(A(n))$ can be constructed by a Turing machine with the input $A(n)$ in a polynomial time. Here for each instance of A we have a special P system which therefore does not need an input. The resulting class of problems is denoted by $\mathbf{MC}^S_X(f)$. Obviously, $\mathbf{MC}_X(f) \subseteq \mathbf{MC}^S_X(f)$ for a given X and a constructible function f.

Particularly, we denote by $\mathbf{PMC}_{\text{div-ne}}$ ($\mathbf{PMC}^S_{\text{div-ne}}$) the class of problems solvable by uniform (semi-uniform, respectively) families of P systems with active membranes in polynomial time. Similarly, we denote by $\mathcal{FAM}_{\text{div-ne}}$ ($\mathcal{FAM}^S_{\text{div-ne}}$) the class of uniform (semi-uniform, respectively) families of these P systems. The abbreviation "div-ne" suggests that a non-elementary membrane division is allowed. The following relations are known [2,12]:

$$\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\text{div-ne}} \subseteq \mathbf{PMC}^S_{\text{div-ne}}. \tag{2}$$

4 RAM simulation of P systems with active membranes

In this section we show that the inclusions reverse to 2 hold, too, i.e. that each (semi-)uniform family of confluent P systems with active membranes can solve only problems in PSPACE. Particularly, we describe how to simulate n steps of any such P systems on a RAM-type computer (and hence also on a Turing machine) in a space polynomial to n.

Consider a membrane system $\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$. For any membrane

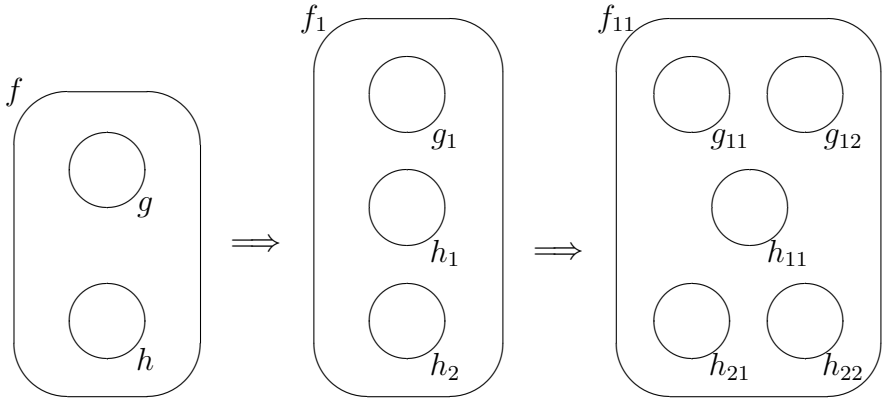

Fig. 2. Indexing of membranes during the first two computational steps.

h of Π, we define its state $S = (M, p)$, where M is the multiset characterizing the content of membrane h and p is its polarization. We use the notation $S.M$ and $S.p$ to refer to these two components of state.

A crucial element of our simulation is the function $State(h, n)$ which computes the state of any membrane h of the system Π after n-th step of computation. If, after n-th step, the membrane h does not exist, the returned value is *nil*. If it is dissolved, the returned value is *dissolved*. Otherwise the functions returns the state $S = (M, p)$ of the membrane.

Our algorithm is described in a high-level language; however, it can be translated into instructions of a RAM computer as defined in e.g. [3]. As such a translation would be easy but cumbersome, it is left to the interested reader.

4.1 Simplified simulation without non-elementary membrane division

We assume without loss of generality that the original labeling of membranes of Π in μ is one-to-one. Hence in the initial configuration the labels identify the membranes uniquely. 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 an index to each membrane label.

In the initial configuration, each index is an empty string. If a membrane is not divided in a computational step, the digit 1 is added to its index. If it is divided using a rule of type (e), the first resulting membrane has added the digit 1 and the second membrane the digit 2 to its index. Hence, after n steps of computation the index of each membrane is an n-tuple of digits from $\{1, 2\}$. Notice that, as in this subsection only elementary membranes can divide, the index of each non-elementary membrane is a string consisting of 1's only. The situation is illustrated at Figure 2.

Now we construct the above mentioned function $State(h_{i_1 i_2 \ldots i_n}, n)$ which computes the state of a membrane $h_{i_1 i_2 \ldots i_n}$ after n computational steps of Π.

(1) If $n = 0$ then return the state of membrane h in the initial configuration and exit.
(2) Calculate recursively $State(h_{i_1 \ldots i_{n-1}}, n-1)$ and store it in the variable S.
(3) If $S = nil$ or $S = dissolved$ then return S and exit.
/ If membrane $h_{i_1 \ldots i_{n-1}}$ did not exist after $(n-1)$ steps, then neither after n steps exists membrane $h_{i_1 i_2 \ldots i_n}$ which could only evolve from $h_{i_1 i_2 \ldots i_{n-1}}$ during step n. */*
(4) Initialize the variable S' which will contain a final state of the membrane after n-th computational step: set $S'.M$ to $\emptyset$ and $S'.p$ to $S.p$.
(5) Initialize auxiliary variables X, X': set $X.M$ and $X'.M$ to $\emptyset$ and set $X.p$ and $X'.p$ to 0.
(6) */* Now we calculate how the membranes embedded in $h_{i_1 \ldots i_{n-1}}$ influence its content at n-th step. */*
For each membrane g contained directly in $h_{i_1 \ldots i_{n-1}}$ calculate recursively $State(g, n-1)$ and store the result in X. Then:
  (a) Try to apply rules of type (a) with parameters g, X, X', X' (see bellow).
  (b) Try to apply rules of type (b) with parameters g, X, X', S. If any rule was applied, skip steps (c) and (d).
  (c) Try to apply rules of type (c) with parameters g, X, X', S'. If any rule was applied, skip step (d).
  (d) Try to apply rules of type (d) with parameters g, X, X', S'.
(7) */* We calculate state of the parent membrane of h. */*
If h is not the skin membrane, then:
  – set $g = Parent(h_{i_1 \ldots i_{n-1}}, n-1)$;
  – calculate recursively $State(g, n-1)$ and store the result to X.
(8) */* Now we simulate evolution of membrane $h_{i_1 \ldots i_{n-1}}$ at step n. */*
  (a) Try to apply rules of type (a) with parameters h, S, S', X.
  (b) Try to apply rules of type (b) with parameters h, S, S', X. If any rule was applied, go to step 9.
  (c) Try to apply rules of type (c) with parameters h, S, S', X. If any rule was applied, go to step 9.
  (d) Try to apply rules of type (d) with parameters h, S, S', X. If any rule was applied, go to step 9.
  (e) If $h_{i_1 \ldots i_{n-1}}$ is an elementary membrane, try to apply rules of type (e) as follows:
    – if $i_n = 1$, then parameters are h, S, S', X;
    – if $i_n = 2$, then parameters are h, S, X, S'.
(9) If $i_n = 2$ and a rule of type (e) was not applied, set S' to nil.
/ If $i_n = 2$, then membrane $h_{i_1 i_2 \ldots i_n}$ could only be created by an application of an (e)-type rule during n-th step. */*
(10) If $S' \neq nil$ and $S' \neq dissolved$ then add the remaining content of $S.M$ to $S'.M$.

(11) Return S' and exit.

The application of rules of the types (a)–(e) is implemented as follows:

Parameters:
h – label of the membrane processed
S – original state of the membrane
S'– final state of the membrane
T – state of another membrane eventually acting at the operation

(a) For each rule $[_h a \to v]_h^\alpha$ in R such that $S.p = \alpha$, remove all the occurrences of a from $S.M$ and add to $S'.M$ the corresponding number of occurrences of v (i.e. of multisets corresponding to v).
(b) For each rule $a[_h\ ]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$ in R such that $S.p = \alpha_1$: if $T.M$ contains a, then remove a from $T.M$, add b to $S'.M$, set $S'.p$ to α_2 and skip all other applicable rules.
(c) For each rule $[_h a\ ]_h^{\alpha_1} \to [_h\ ]_h^{\alpha_2} b$ in R such that $S.p = \alpha_1$: if $S.M$ contains a, then remove a from $S.M$, add b to $T.M$, set $S'.p$ to α_2 and skip all other applicable rules.
(d) For each rule $[_h a\ ]_h^{\alpha} \to b$ in R such that $S.p = \alpha$: if $S.M$ contains a, then remove a from $S.M$, add b to $S'.M$, add $S.M \cup S'.M$ to $T.M$, set S' to *dissolved* and skip all other applicable rules.
(e) For each rule $[_h a\ ]_h^{\alpha_1} \to [_h b\ ]_h^{\alpha_2} [_h c\ ]_h^{\alpha_3}$ in R such that $S.p = \alpha_1$: if $S.M$ contains a, then
– remove a from $S.M$
– add b to $S'.M$ and set $S'.p$ to α_2,
– add c to $T.M$ and set $T.p$ to α_3,
– skip all other applicable rules.

Observe that, with the aid of the function *State*, we can uniquely determine the parent and the children (in terms of the membrane structure tree) of a given membrane $h_{i_1 i_2 \dots i_n}$, without actually storing the membrane structure of Π after n-th step. The function *Parent* can be implemented as follows:

Parameters:
$h_{i_1 i_2 \dots i_n}$ – a membrane whose parent is searched for
n – a number of step

(a) Let g be the parent membrane of h in the initial membrane structure μ. Calculate $State(g_{1\dots1}, n)$.
(b) If the state of $g_{1\dots1}$ was *dissolved* then calculate recursively $Parent(g_{1\dots1}, n)$ and return the result, else return $g_{1\dots1}$.

Similarly, at step 6 we needed to find all children membranes of $h_{i_1 \dots i_{n-1}}$. If g is a child of h in the initial configuration, then each membrane $g_{j_1 j_2 \dots j_n}$,

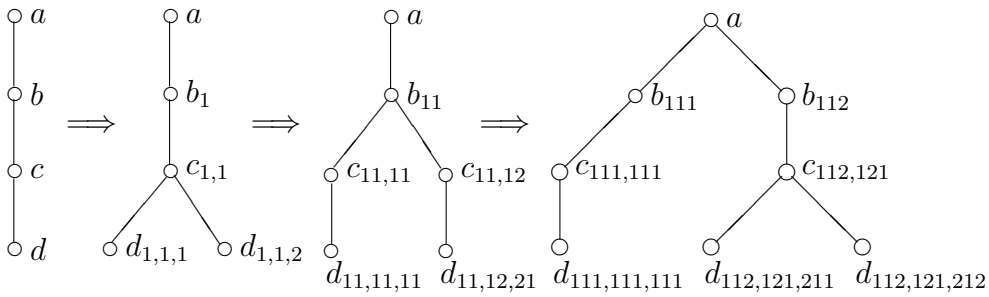

Fig. 3. Example of an indexed membrane structure after three computational steps

with $j_i \in \{1, 2\}$, $1 \leq i \leq n$, is a potential child of $h_{i_1 i_2 \ldots i_n}$. If the state of a child membrane $g_{j_1 j_2 \ldots j_n}$ is *dissolved*, then we have to search recursively the sub-membranes of $g_{j_1 j_2 \ldots j_n}$ until we identify all the non-dissolved children of $h_{i_1 i_2 \ldots i_n}$. Formalization is left to the reader.

Finally, observe that the recursive function *State* was defined correctly because all its recursive calls during the computation of $State(h_{i_1 i_2 \ldots i_n}, n)$ were of the form $State(g_{i_1 i_2 \ldots i_{n-1}}, n-1)$, including the recursive calls during search for parent and children membranes.

4.2 Adding the non-elementary membrane division

When the division of non-elementary membranes is allowed, we first need to introduce an improved indexing of membranes. Unlike the previous simplified case, now in one computational step a division may simultaneously take place at various levels of the membrane structure tree. Therefore, indices are assigned due to the following rules:

(1) The skin membrane has always an empty index.
(2) The index of a membrane at level $k+1$, $k \geq 0$, consists of k tuples of numbers 1 or 2. In the initial configuration each tuple is empty. After n-th computational step each tuple has exactly n elements (becomes an n-tuple).
(3) After each computational step, indices are expanded in the top-down manner. Consider a membrane h at a level $k+1$, $k \geq 0$, with an index $i_{11} \ldots i_{1(n-1)}, \ldots, i_{k1} \ldots i_{k(n-1)}$. If h during step n does not divide, then digit 1 is added to the last $(n-1)$-tuple. If it divides, the resulting two membranes have added 1 and 2, respectively, to their last $(n-1)$-tuples.
(4) Simultaneously the same digit is added to the k-th tuple of indices of all sub-membranes of h (or of the two copies of h if h divides at step n).

The whole situation is illustrated at Figure 3. At the first step, membrane d was divided. At the second step the non-elementary membrane $c_{1,1}$ was divided, each its copy absorbing one of membranes $d_{1,1,1}$ and $d_{1,1,2}$. Finally,

at the third step membrane b_{11} was divided, simultaneously with membrane $d_{11,12,21}$. Observe the following facts:

- An index of a membrane contains indices of all its parent membranes, up to the skin membrane. Simultaneously it contains history of division of the membrane and of all its parent membranes.
- Consider a membrane with index $i_{11} \dots i_{1n}, \dots, i_{k1} \dots i_{kn}$. Then its parent membrane has the index $i_{11} \dots i_{1n}, \dots, i_{(k-1)1} \dots i_{(k-1)n}$, unless it is dissolved.
- A membrane $h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$ has evolved during n-th step from membrane $h_{i_{11}\dots i_{1(n-1)},\dots,i_{k1}\dots i_{k(n-1)}}$.
- 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. Given a membrane $h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$, we can identify its parent membrane and all its potential children membranes (some of them need not exist).

The function *Parent* will in this case look as follows:

Parameters:
$h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$ – a membrane whose parent is searched for
n – a number of step

(a) Let g be the parent membrane of h in the initial membrane structure μ. Calculate $State(g_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}, n)$.
(b) If the state was not *dissolved* then return $g_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$, else calculate recursively $Parent(g_{i_{11}\dots i_{1n},\dots,i_{(k-1)1}\dots i_{(k-1)n}}, n)$ and return the result.

Let us now generalize the function *State* to include non-elementary membrane division. The obvious extension will be that during step 8 also rules of type (f) will be applied. But there is also another more subtle problem.

Unlike the simplified case, the existence of a membrane $h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$ does *not* depends solely on the existence of membrane $h_{i_{11}\dots i_{1(n-1)},\dots,i_{k1}\dots i_{k(n-1)}}$ and on eventual application of (e) or (f)-type rules in this membrane. If any of the upper level membranes containing (recursively) $h_{i_{11}\dots i_{1(n-1)},\dots,i_{k1}\dots i_{k(n-1)}}$ divides using a rule of type (f), then each its sub-membrane is moved into only one of the two resulting membranes. Therefore, the existence of $h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$ depends also on all indices $i_{1n}, \dots, i_{kn}$ and on behavior of all the upper-level membranes. We test this dependence recursively.

More formally, in the description of the function *State* the following steps will be changed:

2. */* We check the existence of membrane $h_{i_{11}\dots i_{1n},\dots,i_{k1}\dots i_{kn}}$ w.r.t. the possible application of type (f) rules in upper level membranes. */*

(i) Initialize a new logical variable L to *false.*

(ii) If $k = 0$ (i.e. h is the skin membrane), go to step (vi).

(iii) Set $g = Parent(h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}, n)$.
Calculate recursively $State(g, n)$.

(iv) If the result is *nil* then return *nil* and exit.
/ If the parent membrane g does not exist, then neither exists its child $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$. */*

(v) If a rule of type (f) was applied during the calculation of $State(g, n)$ in membrane g, set L to *true.* Remember also the polarization values α_3 and α_4 of the rule.

(vi) Calculate recursively $State(h_{i_{11}\ldots i_{1(n-1)},\ldots,i_{k1}\ldots i_{k(n-1)}}, n-1)$ and store it in the variable S.

(vii) If $L = true$ then:
if $S.p = +$ and $i_{(k-1)n} = 2$ or if $S.p = -$ and $i_{(k-1)n} = 1$, then return *nil* and exit.
/ The parent membrane g was divided via a type (f) rule so that its child $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ was moved to its copy different from that specified by $i_{(k-1)n}$. */*

4. Initialize the variable S' which will contain the final state of the membrane after n-th computational step: set $S'.M$ to $\emptyset$ and $S'.p$ to $S.p$.
If $L = true$ then:
– if $S.p = +$ then set $S'.p$ to α_3,
– if $S.p = -$ then set $S'.p$ to α_4.
/ The parent membrane g was divided via a type (f) rule which determines the polarization of its child $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$. */*

8. */* Now we simulate evolution of membrane $h_{i_{11}\ldots i_{1(n-1)},\ldots,i_{k1}\ldots i_{k(n-1)}}$ during n-th step. */*

(a) Try to apply rules of type (a) with parameters h, S, S', X.
If $L = true$ and $S.p \in \{-, +\}$ then go to step 9.
/ The parent membrane is subject to the rule of type (f) at n-th step and hence its child $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ cannot be subject to (b)–(f) type rules in this step. */*

⋮

(e) If $h_{i_{11}\ldots i_{1(n-1)},\ldots,i_{k1}\ldots i_{k(n-1)}}$ is an elementary membrane, try to apply rules of type (e) as follows:
– if $i_{kn} = 1$, then parameters are h, S, S', X;
– if $i_{kn} = 2$, then parameters are h, S, X, S'.

(f) If $h_{i_{11}\ldots i_{1(n-1)},\ldots,i_{k1}\ldots i_{k(n-1)}}$ is a non-elementary membrane, try to apply rules of type (f) as follows:
– if $i_{kn} = 1$, then parameters are h, S, S', X;
– if $i_{kn} = 2$, then parameters are h, S, X, S'.

9. If $i_{kn} = 2$ and neither a rule of type (e) nor (f) was applied, set S' to *nil.*
/ If $i_{nk} = 2$, then membrane $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ could only be created by an application of an (e) or (f) type rule during n-the step. If such a rule was*

not applied, then $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ *does not exist.* */*

Application of rules of type (f) is implemented as follows:

(f) For each rule $[_{h_0}[_{h_1}\,]^{+}_{h_1}\ldots[_{h_k}\,]^{+}_{h_k}\,[_{h_{k+1}}\,]^{-}_{h_{k+1}}\ldots[_{h_n}\,]^{-}_{h_n}]^{\alpha_2}_{h_0}$
$\rightarrow [_{h_0}[_{h_1}\,]^{\alpha_3}_{h_1}\ldots[_{h_k}\,]^{\alpha_3}_{h_k}]^{\alpha_5}_{h_0}\,[_{h_0}[_{h_{k+1}}\,]^{\alpha_4}_{h_{k+1}}\ldots[_{h_n}\,]^{\alpha_4}_{h_n}]^{\alpha_6}_{h_0}$,

such that $S.p = \alpha_2$: if, before application of steps 6(a)–(d), there existed embedded membranes with polarizations + and - (this information can be stored during step 6), then set $S'.p$ to α_5 and $T.p$ to α_6. Skip all other applicable rules.

Again, the recursive function *State* is defined correctly because each recursive call during the computation of $State(h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}, n)$ is in one of the forms

$$State(h_{i_{11}\ldots i_{1n},\ldots,i_{(k-1)1}\ldots i_{(k-1)n}}, n), \tag{3}$$

$$State(h_{i_{11}\ldots i_{1(n-1)},\ldots,i_{k'1}\ldots i_{k'(n-1)}}, n-1), \quad 0 \leq k' \leq d(\mu), \tag{4}$$

where $d(\mu)$ is the depth of the initial membrane structure μ. These two types of calls include also the recursive calls during search for parent and children membranes. One can describe the resulting graph of recursive calls as a two-dimensional lattice with nodes corresponding to pairs (n, k), for $n \geq 0$ and $0 \leq k \leq d(\mu)$, where $d(\mu)$ is depth of the initial membrane structure. An oriented edge from (n, k) to (n', k') denotes a call of $State(h_{i_{11}\ldots i_{1n'},\ldots,i_{k'1}\ldots i_{k'n'}}, n')$ from $State(h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}, n)$.

By 3 and 4, there are edges from each node (n, k) to $(n, k-1)$, if $k > 0$, and to $(n-1, k')$, if $n > 0$, for $0 \leq k' \leq d(\mu)$. Clearly, the graph is acyclic, hence all the recursive calls will be answered after a finite number of steps.

5 Main results

In the previous section we have presented a recursive function *State* which returns a state of any membrane of a P system Π after a given number of computational steps. To employ this function to simulate the computation of a confluent P system with active membranes, we can represent the outer environment surrounding the P system as another region with no applicable rules. Formally, we embed the whole membrane structure μ into a new membrane, say, h_0. Now it remains to subsequently calculate $State(h_0, n)$, for $n = 1, 2, 3, \ldots$, until the object *yes* appears within the content of the new region h_0 or until the computation halts (this we can test by calling $State(h, n)$

for all the membranes h which could potentially exist after n steps, and remembering whether any rule was applied in any of them).

Observe that even if the simulated P system is non-deterministic, the simulation is done in a deterministic way since the applicable rules of all types are browsed always in the same order. Hence, in the case of a non-deterministic but confluent P system, one of all possible computational paths is chosen and the simulation always returns the correct result of computation. Investigating the computational complexity of the function *State*, we obtain the following result:

Theorem 2 $\mathcal{FAM}^{S}_{\text{div-ne}}\text{-TIME}(T^{\mathcal{O}(1)}(n)) \subseteq \text{SPACE}(T^{\mathcal{O}(1)}(n))$

PROOF. Recall first that each P system – member of a semi-uniform family in $\mathcal{FAM}^{S}_{\text{div-ne}}$, starting with an input of size $T^{\mathcal{O}(1)}(n)$, can be also constructed in time $T^{\mathcal{O}(1)}(n)$. Therefore, to prove the inclusion for the whole family, it is enough to prove it for its members of size $T^{\mathcal{O}(1)}(n)$.

We start with determining the space required for computation of $State(h, n)$ which is needed for simulation of n steps of a P system $\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$ of size $n^{\mathcal{O}(1)}$. Let us calculate first the space need to store the content of an arbitrary membrane after n steps of computation. Let

$d(\mu)$ be the depth of the initial membrane structure tree μ,
$p = \max\{|v|; (a \to v) \in R\}$,
$q = \text{card}(V)$,
o_n denote the number of objects within the system after n steps. Hence, $o_0 = |w_1| + \ldots |w_m|$.

By assumption, the values $d(\mu)$, p, q and o_0 are bounded from above by $n^{\mathcal{O}(1)}$. In the rest of the proof we treat them as constants as they are fixed for a given Π. If we considered only the rules of type (a), we obtained $o_n \leq o_0 p^n$. But the membranes can divide, too, and their number after n steps is in bounded by the expression $m(2^{d(\mu)})^n$ (if each membrane at each step divides using a rule of type (e) or (f), which is actually not possible due to conflicts among rules). Hence the total number of the objects is

$$o_n \leq o_0 m(p\ 2^{d(\mu)})^n.$$

As at some step potentially all (but the skin) membranes can dissolve, releasing its content into a single membrane, o_n must be considered also as an upper bound for the number of objects in a single membrane. Then the number of bits necessary to store a content of an arbitrary membrane is

$$s_n \leq q\lceil \log o_n \rceil \leq q\lceil \log(o_0 m) \rceil + nq(\lceil \log p \rceil + d(\mu)) = c_0 + c_1 n \tag{5}$$

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

At each step, the function $State(h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}, n)$ can realize one of the recursive calls of type (3) or (4). During these calls, there must be preserved the following information:

(1) a specification of membrane $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ which requires $kn + \lceil \log m \rceil$ bits, where m is the initial degree of the system Π,
(2) some other variables as L of a constant size c independent on n and k,
(3) only during the calls of type (4): variables S, S', X, X' which store a content of membrane $h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}$ and each of which requires at most s_n bits.

Denote by $S(n,k)$ the space needed to calculate $State(h_{i_{11}\ldots i_{1n},\ldots,i_{k1}\ldots i_{kn}}, n)$, then by (3), (4) and the paragraphs 1–3 above we get the recurrence

$$S(0,k) = s_0, \quad 0 \leq k \leq d(\mu), \tag{6}$$
$$S(n,0) = S(n-1,1) + 4s_n + c, \quad n \geq 1, \tag{7}$$
$$\begin{aligned} S(n,k) = {} & \max\{S(n,k-1), \max\{S(n-1,k') \mid 0 \leq k' \leq d(\mu)\} + 4s_n\} \\ & + nk + c, \quad n \geq 1,\ 1 \leq k \leq d(\mu). \end{aligned} \tag{8}$$

By expanding (8) to a series for $k, k-1, \ldots, 1$ we obtain

$$\begin{aligned} S(n,k) = {} & \max\{S(n,0), \max\{S(n-1,k') \mid 0 \leq k' \leq d(\mu)\} + 4s_n\} \\ & + n\sum_{i=1}^{k} i + kc, \quad n \geq 1. \end{aligned} \tag{9}$$

By denoting

$$S(n) = \max\{S(n,k) \mid 0 \leq k \leq d(\mu)\} \tag{10}$$

and by substituting from (7) we can rewrite the recurrence to the form

$$\begin{aligned} & S(0) = s_0, \\ & S(n) \leq S(n-1) + 4s_n + d(\mu)(d(\mu)+1)n/2 + (d(\mu)+1)c, \quad n \geq 1. \end{aligned}$$

After substituting from (5), a solution to this recurrence is

$$S(n) \leq s_0 + c_2 n^2 + c_3 n$$

for positive constants c_2 and c_3 of size $n^{\mathcal{O}(1)}$, and hence $S(n) = n^{\mathcal{O}(1)}$.

Finally, after substituting n with $T^{\mathcal{O}(1)}(n)$, we get that the simulation of a

$T^{\mathcal{O}(1)}(n)$-time bounded P system Π is $T^{\mathcal{O}(1)}(n)$-space bounded which concludes the proof.

Now, consider the relation (2) and the facts that the second machine class is closed under polynomial time reduction, and that deterministic P systems with active membranes are universal computers [1]. Therefore we can generalize (2) as follows:

$$\text{SPACE}(T^{\mathcal{O}(1)}(n)) \subseteq \mathcal{FAM}_{\text{div-ne}}\text{-TIME}(T^{\mathcal{O}(1)}(n)) \subseteq \mathcal{FAM}^{S}_{\text{div-ne}}\text{-TIME}(T^{\mathcal{O}(1)}(n))$$

Together with Theorem 2 we can conclude that the parallel computation thesis holds for uniform families of confluent P systems with active membranes:

Corollary 3

$$\mathcal{FAM}_{\text{div-ne}}\text{-TIME}(T^{\mathcal{O}(1)}(n)) = \mathcal{FAM}^{S}_{\text{div-ne}}\text{-TIME}(T^{\mathcal{O}(1)}(n)) = \text{SPACE}(T^{\mathcal{O}(1)}(n))$$

Furthermore, as its special case for a polynomial time $T(n)$ we obtain:

Corollary 4

$$\mathbf{PMC}_{\text{div-ne}} = \mathbf{PMC}^{S}_{\text{div-ne}} = \mathbf{PSPACE}.$$

6 Discussion

The results presented in this paper establish an upper bound on the power of confluent P systems with active membranes. However, we do not know the upper bound of the power of their non-deterministic and non-confluent variant. The presented proof cannot be simply extended to this case by using a non-deterministic RAM machine. The reason is that during the simulation of a computation of Π, we did not store its configurations but re-calculated them again and again when needed. Therefore, if these calculations were done non-deterministically, we could get different results in different calculations of the same configuration. Consequently, the whole simulation would not be consistent.

Another variant of P systems with active membranes is presented in [5]. In this variant the non-elementary membrane division is not performed by rules of type (f) above. Instead, rules of the form $[_h a\,]_h^{\alpha_1} \rightarrow [_h b\,]_h^{\alpha_2}[_h c\,]_h^{\alpha_3}$ are used for both elementary and non-elementary membrane division. If in the membrane h there are other objects than a and any embedded membranes, then during the same step they may evolve in a usual way and then the results of this evolution are copied into both copies of membrane h.

Eventual simulation of this variant of P systems with active membranes would follow almost exactly the way described in Section 4.1. The only differences would be that also non-elementary membranes can divide and that the structural indexing of the type $i_{11} \dots i_{1n}, \dots, i_{k1} \dots i_{kn}$ would be used. Therefore, we claim that Theorem 2 holds also in this case, and the class of problems solvable in a polynomial time by this variant of P systems is bounded from above by **PSPACE**. However, there is no formal proof yet that these P systems can also solve PSPACE-complete problems in polynomial time. Hence, the validity of parallel computation thesis (3) in this case remains open. Another open problem is the (non-)validity of the results contained in this paper in the case of P systems with active membranes and *minimal parallelism* [5].

Acknowledgements

Research was partially supported by the Czech Science Foundation, grant No. 201/06/0567, and by the Programa Ramón y Cajal, Ministerio de Ciencia y Tecnología, Spain.

References

[1] A. Alhazov, R. Freund, A. Riscos-Núñez, One and two polarizations, membrane creation and objects complexity in P systems, in: G. Ciobanu, Gh. Păun (Eds.), Technical Report 05-11, Institute e-Austria, Timişoara, Romania, First International Workshop on Theory and Application of P Systems (TAPS), 2005, pp. 9–18.

[2] A. Alhazov, C. Martin-Vide, L. Pan, Solving a PSPACE-complete problem by P systems with restricted active membranes, Fundamenta Informaticae, 58, 2 (2003), 67–77.

[3] J.L. Balcazar, J. Diaz, J. Gabarro, Structural Complexity I, Second Edition, Springer-Verlag, Berlin, 1995.

[4] J.L. Balcazar, J. Diaz, J. Gabarro, Structural Complexity II, Springer-Verlag, Berlin, 1991.

[5] G. Ciobanu, L. Pan, G. Păun, M.J. Pérez-Jiménez, P Systems with Minimal Parallelism, submitted.

[6] L.M. Goldschlager, A universal interconnection pattern for parallel computers, J. Assoc. Comput. Mach., 29 (1982), 1073–1086.

[7] Gh. Păun, Computing with Membranes, J. Comput. System Sci., 61 (2000), 108–143.

[8] Gh. Păun, Membrane Computing: an Introduction, Springer-Verlag, Berlin, 2002.

[9] Gh. Păun, P systems with active membranes: attacking NP complete problems, J. Automata, Languages and Combinatorics, 6, 1 (2001), 75–90.

[10] M.J. Pérez-Jiménez, A.R. Jiménez, F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, Natural Computing, 2 (2003), 265–285 .

[11] P. Pudlák, Complexity theory and genetics: The computational power of crossing-over. Information and Computation, 171 (2001), 201–223.

[12] P. Sosík, The computational power of cell division: beating down parallel computers? Natural Computing, 2–3 (2003), 287–298.

[13] P. van Emde-Boas, The second machine class: models of parallelism, in: J. van Leeuwen, J.K. Lenstra and A.H.G. Rinnooy Kan (Eds.), Parallel Computers and Computations, CWI Syll., Centre for Mathematics and Computer Science, Amstedam, 1985.

[14] J. van Leeuwen, J. Wiedermann, Array processing machines, in: L. Budach (Ed.), Fundamentals of Computational Theory 1985, Cottbus GDR, Springer-Verlag, LNCS 199 (1985), pp. 257–268.