Membrane computing and complexity theory: 
A characterization of PSPACE

Petr Sosík a, b, *, Alfonso Rodríguez-Patón a

a Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain
b Institute of Computer Science, Silesian University, 74601 Opava, Czech Republic

Received 29 May 2006; received in revised form 3 October 2006
Available online 7 November 2006

Abstract

A P system is a natural computing model inspired by information processing in cells and cellular membranes. We show that confluent P systems with active membranes solve in polynomial time exactly the class of problems PSPACE. Consequently, these P systems prove to be equivalent (up to a polynomial time reduction) to the alternating Turing machine or the PRAM computer. Similar results were achieved also with other models of natural computation, such as DNA computing or genetic algorithms. Our result, together with the previous observations, suggests that the class PSPACE provides a tight upper bound on the computational potential of biological information processing models.

Keywords: Biological computation; P system; PSPACE; Alternating Turing machine

1. Introduction

Membrane systems, called also P systems, are bio-inspired computing models belonging to a broader family of so-called biological or natural computing. Besides established natural computing topics like artificial neural networks, genetic algorithms, ant algorithms, DNA computing etc., P systems are trying to capture computational aspects of cell metabolism and information interchange. Particularly, they focus on selective particle recognition by membranes, controlled transport through protein channels, cell metabolism or membrane division and dissolution. These processes are modeled in P systems by means of parallel multiset processing in separate cell-like regions. The aim of these models is to identify operations which give to cellular systems their information-processing strength and to prepare their possible implementation in vitro or in silico.

This motivation is complemented by recent successful attempts to use P systems as models of biological phenomena, especially in situations with a low molecule concentration. Preliminary results can be found in [5], modeling phenomena as mechanosensitive channels, bacteria respiration, photosynthesis or the p53 signaling pathways.

* Expanded version of a paper presented at the 12th Int. Meeting on DNA Computing (DNA 12).
* Corresponding author.
E-mail addresses: petr.sosik@fpf.slu.cz (P. Sosík), arpaton@fi.upm.es (A. Rodríguez-Patón).

022-0000/$ – see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2006.10.001
The bibliography of P systems started with seminal paper [10]. Since then, P systems became a dynamically growing research field, selected in 2003 by ISI as a fast “Emerging Research Front” in Computer Science (http://esi-topics.com/erf/october2003.html). For an introduction and overview of P systems, we refer to [11], while a complete and up-to-date bibliography can be found online at [19].

Among the most powerful models of P systems one can mention P systems with active membranes and P systems with membrane creation. Here we investigate the former model, introduced in [12] and studied, e.g., in [1,4,7,8,13,14,16,18] and other papers; see [19] for a complete list. The main ingredients of this model are the polarization, division and dissolution of membranes. Interestingly enough, by imposing simple restrictions on these operations one obtains a scale of computational power corresponding to elementary complexity classes. For instance, without the possibility of membrane division these P systems can solve in polynomial time the class of problems P [8]. Rather surprisingly, the same result is obtained when preserving membrane division but removing membrane polarization and dissolution [7]. When only elementary membrane division is allowed, the resulting class contains both NP and coNP [13,18]. Therefore, the open question of the computational power of membrane division is closely related to the P ̸= NP problem. Finally, when no restrictions are imposed on the model, it can solve the PSPACE-complete problem QSAT in a polynomial time [1,16]. We refer the reader to Section 3 for formal specification of these results.

In this paper a characterization of the power of confluent P systems with active membranes is given. We show that the class of problems they solve in polynomial time is equal to PSPACE. Consequently, these P systems satisfy the Parallel Computation Thesis [17]:

\[ M \text{-PTIME} = M \text{-NP TIME} = \text{PSPACE}, \]

where M-(N)PTIME is the class of problems solved in polynomial time by a (non-)deterministic machine M. We recall that computers satisfying (1) form the second machine class, whose members are the alternating Turing machine, SIMDAG (also known as SIMD PRAM) and other parallel models [17].

This result is related to recent studies of other natural computing models. For example, in the field of DNA computing, analogous result was given in [3]. Later [6] presented another, more robust DNA computing model capturing PSPACE in polynomial time. Also, when abstracting from biochemical aspects of DNA to the operation of genetic crossing-over, one obtains a second-class computer embodied in so-called genetic Turing machine ([15], see also [2]). Altogether, these studies suggest that the class PSPACE provides the tight upper bound on the computational potential of natural computing machinery.

2. Definitions

In this section we give a brief description of P systems with active membranes due to [11,12], where more details can also 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 [12] illustrates the situation: the membrane structure in Fig. 1 is identified by the string


Fig. 1. A membrane structure and its associated tree.
In what follows, we interpret the membrane structure of $\Pi$ as a rooted tree and refer occasionally to its elements—membranes as nodes in this tree. The 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 $[i]_0^+$, $[i]_0^-$, $[i]_0^0$ in the three cases, respectively.

The membranes delimit regions, precisely identified by the membranes. 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 over an alphabet $V$ can be represented by a string $x \in V^*$ (by $V^*$ we denote the free monoid generated by $V$ with respect to the concatenation and the identity $\lambda$).

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) $\mu$ is a membrane structure, consisting of $m$ membranes, labeled (not necessarily in a one-to-one manner) with elements of $H$; all membranes in $\mu$ are supposed to be neutral;
(v) $w_1, \ldots, w_m$ are strings over $V$, describing the multisets of objects placed in the regions of $\mu$;
(vi) $R$ is a finite set of developmental rules, of the following forms:

(a) $[h a \rightarrow v]^\alpha_h$,
   for $h \in H$, $\alpha \in \{+, -, 0\}$, $a, v \in V^*$
   (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) $a[h]^\alpha_h \rightarrow [h b]^\alpha_h$,
   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) $[h a]^\alpha_h \rightarrow [h b]^\alpha_h [h c]^\alpha_h$,
   for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in V$
   (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) $[h a]^\alpha_h \rightarrow b$,
   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) $[h a]^\alpha_h \rightarrow [h b]^\alpha_h [h c]^\alpha_h$,
   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) $[h_0 h_1]^{+\alpha_1} \cdots [h_k]^{+\alpha_k} [h_{k+1}]^{-\alpha_k} \cdots [h_n]^{-\alpha_k} \rightarrow [h_0 h_1]^{+\alpha_3} [h_2]^{+\alpha_5} [h_0 h_{k+1}]^{+\alpha_4} [h_0 h_n]^{+\alpha_6}$,
   for $n > k \geq 1$, $h_i \in H$, $0 \leq i \leq n$, and $\alpha_2, \alpha_4, \alpha_6 \in \{+, -, 0\}$
   (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; all membranes of opposite polarizations are always separated by applying this rule; if the membrane labeled $h_0$ 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 subject to only one rule of type (a)–(e) and a membrane \( h \) can be subject to only one rule of type (b)–(f). In the case of type (f) rules, this means that none of the membranes \( h_0, \ldots, h_n \) listed in the rule can be simultaneously subject to another rule of type (b)–(f). However, this restriction does not apply to membranes with neutral charge contained in \( h_0 \). In general, an application of the rules is performed as follows:

(i) in every step, first the rules are assigned to objects and membranes in a maximal way (any object and membrane which can evolve by a rule of any form, should evolve), and then all the rules are simultaneously applied;
(ii) all objects and membranes which cannot evolve pass unchanged to the next step;
(iii) 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;
(iv) 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 \( \Pi \) at a given moment, together with all multisets of objects contained in its regions, form the configuration of the system. The \((m+1)\)-tuple \((\mu, w_1, \ldots, w_m)\) is the initial configuration. We can pass from one configuration to another 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. It was shown in [12] that in such a way P systems with active membranes can generate any recursively enumerable set and hence is computationally universal.

Example. Let \( \Pi = (V, H, \mu, w_0, \ldots, w_n, R) \) be a P system with active membranes, where:

- \( V = \{b_0\} \cup \{b_i, f_i, t_i | 1 \leq i \leq n\} \);
- \( H = \{0, 1, \ldots, n\} \);
- \( \mu = \{0|1| \ldots |n|_{n-1}\}_{1}^{0} \);
- \( w_0 = w_1 = \cdots = w_{n-1} = \lambda, \ w_n = b_0 \);
- \( R \) contains the following rules:
  (i) \([n] b_i [n] \rightarrow [n] b_{i+1} [n]_{i+1}, \ 0 \leq i \leq n - 1,\)
  (ii) \([n] b_i \rightarrow b_{i+1} f_i [n]_{i+1}^{0}, \ [n] b_i \rightarrow b_{i+1} t_i [n]_{i+1}^{0}, \ 0 \leq i \leq n - 1,\)
  (iii) \([i]_{i-1} b_i [i]_{i-1} \rightarrow [i]_{i-1} b_i [i]_{i-1}^{0}, \ [i]_{i-1} b_i [i]_{i-1}^{0}, \ 2 \leq i \leq n,\)

Computation of the system \( \Pi \) is illustrated in Fig. 2. In the first step the elementary membrane at level \( n \) is divided by rules (i) into two parts. In the second step, symbols \( f_1 \) and \( t_1 \) are produced in the two resulting membranes by rule (ii).

![Fig. 2. An example—expansion of a simple membrane structure into a binary tree.](image-url)
Simultaneously rule (iii) divides the non-elementary membrane $[i−1]_0^{i−1}$ into two. This cycle is repeated $n$ times and waves corresponding to the division by rule (iii) climb up the membrane tree towards its root. The computation stops after $2n − 1$ steps when the membrane structure forms a balanced binary tree. Each of its $2^n$ leafs contains a set $\{x_1, x_2, \ldots, x_n, \epsilon_n\}$, where $x_i \in \{f_i, t_i\}$, $1 \leq i \leq n$, such that all possible $n$-tuples are present.

In this paper we study the accepting variant of P systems which solves decision problems. A distinguished region contains, at the beginning of the computation, an input—a description of an instance of a problem. The result of the computation (a solution to the instance) is “yes” if a distinguished object yes is expelled during the computation, otherwise the result is “no.” Such a P system is called deterministic if for each input a unique sequence of configurations exists. A P system is called confluent if it always halts and, starting from the same initial configuration, it always returns the same result “yes” or “no.” Therefore, given a fixed initial configuration, a confluent P system can non-deterministically choose from various sequences of configurations, but all of them must lead to the same result.

3. Complexity classes of P systems

Consider a decision problem $X$, i.e. a set of instances $\{x_1, x_2, \ldots\}$ such that to each $x_i$ there is an unique answer “yes” or “no.” A typical situation in many biocomputing models is that each “machine” (e.g. a system of DNA molecules and reactions, or a P system) can solve only a limited number of instances. However, the problem $X$ can still be solvable by a given class of devices in such a sense that each instance is solved by some class member. Hence it is more natural to consider families of P systems for solving computational problems. The same approach is used in the computational complexity theory when dealing with finite-state machines such as boolean circuits. In this manner there have been defined complexity classes for P systems [13].

We denote by $|x_i|$ the size of an instance $x_i$ of a problem $X$. In a usual representation $x_i$, $i = 1, 2, \ldots$, are words over a fixed finite alphabet and $|x_i|$ is the length of $x_i$.

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

(i) There exists a uniform family of P systems $\Pi_X = (\Pi_X(1); \Pi_X(2); \ldots)$ of type $D$: each $\Pi_X(n)$ can be constructed by a deterministic Turing machine with input $n$ in a time polynomial to $n$.

(ii) 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.”

(iii) 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.”

(iv) $\Pi_X$ is $f$-efficient: $\Pi_X(n)$ always halts in at most $f(n)$ steps.

By “proper encoding” we mean that it is computable by a deterministic Turing machine in a polynomial time. 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 $\mathbf{MC^S_D}(f)$. Obviously, $\mathbf{MC_D}(f) \subseteq \mathbf{MC^S_D}(f)$ for a given class $D$ and a constructible function $f$.

Particularly, we denote by

$$\mathbf{PMC_D} = \bigcup_{k \in \mathbb{N}} \mathbf{MC_D}(O(n^k)),$$

$$\mathbf{PMC^S_D} = \bigcup_{k \in \mathbb{N}} \mathbf{MC^S_D}(O(n^k)),$$

the classes of problems solvable by uniform (semi-uniform, respectively) families of P systems in polynomial time.

We also denote by $\mathbf{AM}$ ($\mathbf{EAM}$, $\mathbf{NAM}$) the classes of P systems with active membranes (with elementary membrane division only and without membrane division, respectively). The following relations are known [1,7,13,14,16, 18]:

$$\mathbf{P} = \mathbf{PMC_NAM}.$$
\[ P = \text{PMC}_{\mathcal{A}M,n\delta,n\pi}, \quad (3) \]

\[ \text{NP} \cup \text{coNP} \subseteq \text{PMC}_{\mathcal{E}AM}, \quad (4) \]

\[ \text{PSPACE} \subseteq \text{PMC}_{\mathcal{A}M} \subseteq \text{PMC}_{\mathcal{S}AM}^S, \quad (5) \]

\[ \text{PMC}_{\mathcal{A}M} \subseteq \text{EXPTIME}. \quad (6) \]

where in (3) we denote by \( \mathcal{A}M, n\delta, n\pi \) the class of P systems with active membranes, without membrane dissolution and polarization.

4. The characterization of PSPACE

In this section we show that inclusions reverse to (5) hold as well. We employ 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. In this way a result of any \( T(n) \)-time-bounded computation of a confluent accepting P system with active membranes can be found in a space polynomial to \( T(n) \).

Notice also that the simulated P system is confluent (hence possibly non-deterministic), but its simulation is deterministic—the rules in membranes are always applied in the same order. Hence we simulate only one possible sequence of configurations of the P system. This 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 (a), (b), (c), (d), (e), (f), in this order. The confluency condition ensures that such a simulation always leads to a correct result.

**Theorem 2.** \( \text{PMC}_{\mathcal{A}M}^S \subseteq \text{PSPACE} \).

**Proof.** Consider a membrane system \( \Pi = (V, H, \mu, w_1, \ldots, w_m, R) \). For any membrane \( h \) of \( \Pi \), we define its state \( S = (M, p) \), where \( M \) is the multiset characterizing the contents 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 construction is the function \( \text{State} \) which computes the state of any membrane \( h \) of \( \Pi \) at a given step of computation. The key observation is that this state can be computed from the state of \( h \), state of its parent membrane and states of all its (recursively) embedded membranes at the previous step. The basic strategy is the following:

- verify whether the predecessor of our membrane \( h \) existed at all at the previous computational step;
- calculate recursively the state of this predecessor at the end of the previous step;
- calculate recursively previous states of all the membranes embedded in \( h \), then enumerate objects which \( h \) sends/receives to/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 (a)–(f) in membrane \( h \) during the given step.

- **Simplified simulation without non-elementary membrane division**

We assume without loss of generality that the original labeling of membranes of \( \Pi \) in \( \mu \) is one-to-one. However, during the computation of \( \Pi \) 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.

In the initial configuration, each index is an empty string. If a membrane is not divided in a computational step, then digit 1 is attached to its index. If a membrane is divided using a rule of type (e), the first resulting membrane has attached 1 and the second membrane 2 to its index. Hence, after \( n \) steps of computation the index of each membrane is an \( n \)-tuple of digits from \( \{1, 2\} \). In this way we assign an unique index to each membrane which could potentially exist if all elementary membranes divide at every step. In a particular computation of \( \Pi \) this need not hold, and hence
some indices may denote non-existing membranes. Notice also that since we now consider only division of elementary membranes, the index of each non-elementary membrane consists solely of 1’s. The situation is illustrated in Fig. 3.

Now we construct the above-mentioned function $\text{State}(h[i_1i_2\ldots i_n], n)$ which computes the state of a membrane $h[i_1i_2\ldots i_n]$ after $n$ computational steps of $\Pi$. If the membrane $h[i_1i_2\ldots i_n]$ has been dissolved, the returned value is dissolved. If it does not exist after $n$ steps, the returned value is nil. The second argument $n$ might seem redundant as it is implicitly contained in the index of $h$. However, it will be needed in the second part of the proof.

**Function State**
/* Returns the state of a membrane $h[i_1i_2\ldots i_n]$ after $n$ computational steps. */

**Parameters:** /* in all the functions passed by reference. */
$h[i_1i_2\ldots i_n], n$.

**Local variables:** $S, S', X, X', \text{Elementary}$.

(i) If $n = 0$ then return the state of membrane $h$ in the initial configuration and exit.
(ii) $S$ ← $\text{State}(h[i_1\ldots i_{n-1}], n - 1)$.
(iii) If $S = \text{nil}$ then return nil and exit.
   /* If membrane $h[i_1 \ldots i_{n-1}]$ did not exist after $(n - 1)$ steps, then after $n$ steps its successor $h[i_1i_2\ldots i_n]$ cannot exist as well. */
(iv) If $S = \text{dissolved}$ then:
   if $i_n = 2$ then return nil, else return dissolved, and exit.
   /* If membrane $h[i_1 \ldots i_{n-1}]$ has been dissolved during the first $(n - 1)$ steps of $\Pi$, then it cannot divide during step $n$. */
(v) $S'.M$ ← $\emptyset$, $S'.p$ ← $S.p$, $\text{Elementary}$ ← true.
   /* $S'$ will contain the final state of membrane $h$ after the $n$th step of $\Pi$. */
(vi) $\text{Contribution_from_children}(h[i_1 \ldots i_n], S, S', \text{Elementary})$.
   /* We calculate the contribution to the state of $h[i_1 \ldots i_n]$ from its embedded membranes due to possible application of rules (b), (c), (d) in these membranes. */
(vii) $X.M$ ← $\emptyset$, $X'.M$ ← $\emptyset$, $X.p$ ← 0, $X'.p$ ← 0.
(viii) /* We calculate the state of the parent membrane of $h$. */
   If $h$ is not the skin membrane, then $X$ ← $\text{State}(\text{Parent}(h[i_1 \ldots i_{n-1}]), n - 1)$.
(ix) /* Now we simulate the evolution of membrane $h[i_1 \ldots i_{n-1}]$ at step $n$. */
   (a) $\text{Try_rules_a}(h, S, S', X)$.
   (b) $\text{Try_rules_b}(h, S, S', X)$; if any rule was applied, go to step (x).
   (c) $\text{Try_rules_c}(h, S, S', X)$; if any rule was applied, go to step (x).
   (d) $\text{Try_rules_d}(h, S, S', X)$; if any rule was applied, go to step (x).
(e) If Elementary then:
  – if \( i_n = 1 \), then \( \text{Try\_rules\_e}(h, S, S', X) \), else \( \text{Try\_rules\_e}(h, S, X, S') \).

(x) If \( i_n = 2 \) and a rule of type (e) was not applied, then \( S' \leftarrow \text{nil} \).
  /* 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 at n-th step. */

(xi) If \( S' \neq \text{nil} \) and \( S' \neq \text{dissolved} \) then \( S'.M \leftarrow S'.M \cup S.M \).

(xii) Return \( S' \) and exit.

Procedures \( \text{Try\_rules\_a} - \text{Try\_rules\_e} \)
/* Apply a given type of rules contained in a membrane \( h \); remove their left-hand sides from its initial state \( S \) and add their right-hand sides to its final state \( S' \). */

Parameters:
\( h \)—label of the membrane processed,
\( S \)—initial state of the membrane,
\( S' \)—final state of the membrane,
\( T \)—state of another membrane eventually acting at the operation.

(a) For each rule \( [h a \rightarrow v]^\alpha_h \) in \( R \) such that \( S.p = \alpha \):
  – remove all the occurrences of \( a \) from \( S.M \),
  – add to \( S'.M \) the same number of multisets represented by \( v \).

(b) For each rule \( a[h]^\alpha_h \rightarrow [h a]^\alpha_h \) in \( R \): if \( S.p = \alpha \) and \( a \in T.M \) then:
  \( T.M \leftarrow T.M \setminus \{a\} \), \( S'.M \leftarrow S'.M \cup \{b\} \), \( S'.p \leftarrow \alpha_2 \), and skip all other applicable rules.

(c) For each rule \( [h a]^\alpha_h \rightarrow [h a]^\alpha_h \) in \( R \): if \( S.p = \alpha \) and \( a \in S.M \) then:
  \( S.M \leftarrow S.M \setminus \{a\} \), \( T.M \leftarrow T.M \cup \{b\} \), \( S'.p \leftarrow \alpha_2 \), and skip all other applicable rules.

(d) For each rule \( [h a]^\alpha_h \rightarrow b \) in \( R \): if \( S.p = \alpha \) and \( a \in S.M \) then:
  \( S.M \leftarrow S.M \setminus \{a\} \), \( S'.M \leftarrow S'.M \cup \{b\} \), \( T.M \leftarrow T.M \cup S.M \cup S'.M \), \( S' \leftarrow \text{dissolved} \), and skip all other applicable rules.

(e) For each rule \( [h a]^\alpha_h \rightarrow [h a]^\alpha_h \) in \( R \): if \( S.p = \alpha \) and \( a \in S.M \) then:
  \( S.M \leftarrow S.M \setminus \{a\} \),
  \( S'.M \leftarrow S'.M \cup \{b\} \), \( S'.p \leftarrow \alpha_2 \),
  \( T.M \leftarrow T.M \cup \{c\} \), \( T.p \leftarrow \alpha_3 \),
  and skip all other applicable rules.

Observe that, with the aid of function \( \text{State} \), we can uniquely determine parent and children (in terms of the membrane structure tree) of a given membrane \( h[i_1 i_2 \ldots i_n] \), without actually storing the membrane structure of \( \Pi \) after n-th step.

Function \( \text{Parent} \)
/* Returns the parent membrane of \( h[i_1 i_2 \ldots i_n] \). Never called for the skin membrane. */

Parameters:
\( h[i_1 i_2 \ldots i_n] \)—a membrane whose parent is searched for

(i) Let \( g \) be the parent membrane of \( h \) in the initial membrane structure \( \mu \).
(ii) If \( \text{State}(g[1 \ldots 1], n) = \text{dissolved} \) then return \( \text{Parent}(g[1 \ldots 1]) \), else return \( g[1 \ldots 1] \).

Procedure \( \text{Contribution\_from\_children} \)
/* Calculates the interaction of a membrane \( h[i_1 \ldots i_n] \) with its children membranes at step \( n \) by:
  – sending objects into children membranes by rules of type (b),
  – receiving objects from children membranes by rules of type (c),
  – receiving contents of children membranes by rules of type (d).
Furthermore, it tests whether the membrane \( h[i_1 \ldots i_{n-1}] \) was elementary. */
Parameters:
\[ h[i_1 \ldots i_n] \]—a membrane to which its children contribute,
\[ S \]—set of objects in \( h \) which have not been yet subject to any rule,
\[ S' \]—set of objects which are produced in \( h \) as a result of rule applications,
\[ \text{Elementary} \]—an output parameter indicating whether \( h[i_1 \ldots i_{n-1}] \) was an elementary membrane.

Local variables: \( X, X', g, \text{Elementary}' \).

For each children membrane \( g \) of \( h \) in the initial membrane structure \( \mu \), and for each index \( j_1 \ldots j_n \) such that \( j_i \in \{1, 2\}, 1 \leq i \leq n \), repeat:

(i) \( X \leftarrow \text{State}(g[j_1 \ldots j_{n-1}], n-1) \), \( X' \leftarrow (\emptyset, 0) \).
(ii) If \( X = \text{nil} \) then skip the remaining steps.
(iii) If \( X = \text{dissolved} \) then:
- \( \text{Contribution_from_children}(g[j_1 \ldots j_n], S, S', \text{Elementary}) \),
- skip the remaining steps.
/* If membrane \( g[j_1 \ldots j_{n-1}] \) is dissolved, then its children are actually children of \( h[i_1 \ldots i_{n-1}] \). */
(iv) \( \text{Elementary} \leftarrow \text{false} \).
(v) \( \text{Elementary}' \leftarrow \text{true} \).
(b) \( \text{Contribution_from_children}(g[j_1 \ldots j_n], X, X', \text{Elementary}') \).
/* We recursively calculate the contribution of children membranes to \( g[j_1 \ldots j_n] \) for the case it is dissolved at \( n \)-th step and releases its contents to \( h[i_1 \ldots i_n] \). */
(vi) \text{Try_rules}_a(g, X, X', X').
(vii) \text{Try_rules}_b(g, X, X', S); if any rule was applied, skip the remaining steps.
(viii) \text{Try_rules}_c(g, X, X', S'); if any rule was applied, skip the remaining steps.
(ix) \text{Try_rules}_d(g, X, X', S').
/* If \( g[j_1 \ldots j_n] \) is dissolved, then its recursively calculated contents (including the contents of eventual lower-level dissolved membranes) is added to \( S' \). */

Notice that we had to try an application of all types of rules in the child membrane \( g[j_1 \ldots j_n] \) even the rules of type (a) cannot contribute to the contents of \( h[i_1 \ldots i_n] \). The reason is that to keep the simulation deterministic, we must simulate the application of rules in all the procedures in the same order.

Observe also that the recursive function \( \text{State} \) is defined correctly because all its recursive calls during the computation of \( \text{State}(h[i_1 i_2 \ldots i_n], n) \) are of the form \( \text{State}(g[j_1 j_2 \ldots j_{n-1}], n-1) \), i.e. referring to a state of (another) membrane at the previous step. The same holds for the recursive calls of \( \text{State} \) in procedures \( \text{Parent} \) and \( \text{Contribution_from_children} \).

- **Adding the non-elementary membrane division**

  When the division of non-elementary membranes is allowed, we first need to refine the 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:

  (i) The skin membrane always has an empty index.
  (ii) 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.
  (iii) After each computational step, indices are extended in a top-down manner. Consider a membrane \( h[i_1 \ldots i_{(a-1)}, \ldots i_{k1} \ldots i_{kn(n-1)}] \). 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.
  (iv) Simultaneously the same digit is attached to the \( k \)th tuple of indices of all sub-membranes of \( h \).
The whole situation is illustrated in Fig. 4. At the first step, membrane $d$ was divided. At the second step, the non-elementary membrane $c_{1,1}$ was divided, each of its copies 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 $d_{11,11,21}$. Observe the following facts:

- An index of a membrane contains as prefixes the indices of all its parent membranes, up to the skin membrane.
- The parent membrane of $h[i_{11} \ldots i_n, \ldots, i_{k1} \ldots i_{kn}]$ has the index $i_{11} \ldots i_n, \ldots, i_{(k-1)n}$, unless it is dissolved.
- A membrane $h[i_{11} \ldots i_n, \ldots, i_{k1} \ldots i_{kn}]$ evolved from membrane $h[i_{11} \ldots i_{(n-1)}, \ldots, i_{k1} \ldots i_{(n-1)}]$ at the $n$th step of computation.
- Given an initial membrane structure $\mu$ and a number $n \geq 0$, we can effectively enumerate all the membranes which could potentially exist in $\mu$ after $n$ steps.
- Given a membrane $h[i_{11} \ldots i_n, \ldots, i_{k1} \ldots i_{kn}]$, we can identify its parent membrane and all its potential children membranes (some of them need not exist).

Instead of the detailed notation of indices $i_{11} \ldots i_n, \ldots, i_{k1} \ldots i_{kn}$ we will in the sequel use also its shorter version $i_{11} \ldots i_{kn}$. Below we give modified versions of the recursive functions simulating computation of a P system.

**Procedure Parent**

**Parameters:**
- $h[i_{11} \ldots i_{kn}]$—a membrane whose parent is searched for,
- $X$—state of the parent membrane,
- $L$—logical value “a rule of type (f) was applied,”
- $\alpha_3, \alpha_4$—polarization values of the applied rule.

(i) Let $g$ be the parent membrane of $h$ in the initial membrane structure $\mu$.
(iii) If a rule of type (f) was applied during the computation of $X$ in membrane $g[i_{11} \ldots i_{(k-1)n}]$, then $L \leftarrow true$, else $L \leftarrow false$.
(v) If $X = dissolved$ then return $\text{Parent}(g[i_{11} \ldots i_{(k-1)n}], X, L, \alpha_3, \alpha_4)$, else return $g[i_{11} \ldots i_{(k-1)n}]$.

**Procedure Contribution_from_children**

**Parameters:**
- $h[i_{11} \ldots i_{kn}]$—a membrane to which its children contribute,
- $S$—a set of objects in $h$ which have not yet been subject to any rule,
- $S'$—a set of objects which are produced in $h$ as a result of rule applications,
- $h[i_{11} \ldots i_{kn}]$ is an elementary membrane,
- $Nplus, Nminus$—multisets of children membranes in $h[i_{11} \ldots i_{kn}]$ with polarization $+$ or $-$. 
Local variables: X, X’, Y, g, Lplus, Lminus, Elementary’.

For each children membrane g of h in the initial membrane structure μ, and for each n-tuple \( j_{(k+1)_1} \ldots j_{(k+1)_n} \), \( f_{(k+1)_1} \in \{1, 2\}, 1 \leq \ell \leq n: 

(i) \( X \leftarrow \text{State}(g[i_{11} \ldots i_{(n-1)_1}, j_{(k+1)_1} \ldots j_{(k+1)_{(n-1)_1}}], n - 1), X' \leftarrow (\emptyset, 0). \)

(ii) If \( X = \text{nil} \) then skip the remaining steps.

(iii) If \( X = \text{dissolved} \) then:
- Contribution_from_children(g[i_{11} \ldots i_{kn}, j_{(k+1)_1} \ldots j_{(k+1)_n}], S, S’, Elementary, Nplus, Nminus),
- skip the remaining steps.

(iv) Elementary ← false.
/* Now we know that membrane h[i_{11} \ldots i_{kn}] contains at least one children membrane. */

(v) Elementary’ ← true, Lplus ← \emptyset, Lminus ← \emptyset.

(vi) Contribution_from_children(g[i_{11} \ldots i_{kn}, j_{(k+1)_1} \ldots j_{(k+1)_n}], X, X’, Elementary’, Lplus, Lminus).
/* Calculate recursively the contribution of children membranes to g[i_{11} \ldots i_{kn}, j_{(k+1)_1} \ldots j_{(k+1)_n}] for the case g is dissolved at n-th step and releases its content to h[i_{11} \ldots i_{kn}]. */

(vii) Y ← (\emptyset, 0).

(viii) Try_rules_a(g, X, X’, X’).

(ix) Try_rules_b(g, X, X’, S); if any rule was applied, go to step (xiv).

(x) Try_rules_c(g, X, X’, S’); if any rule was applied, go to step (xiv).

(xi) Try_rules_d(g, X, X’, S’); if any rule was applied, go to step (xiv).

(xii) If Elementary’ then Try_rules_e(g, X, X’, Y).

(xiii) If not Elementary’ then Try_rules_f(g, X, X’, Y, Lplus, Lminus).

(xiv) If X.p ≠ 0 then:
- if a rule (b)–(f) was applied, then Nplus ← nil, Nminus ← nil,
- else if X.p = + then Nplus ← Nplus \cup \{g\},
- else if X.p = – then Nminus ← Nminus \cup \{g\}.
/* If g is polarized + or – and is subject to a rule of type (b)–(f), then its parent h cannot be simultaneously subject to a rule of type (f). This is indicated by the constant nil. */

Let us now generalize the function State to include the non-elementary membrane division. Unlike the simplified case, now the existence of a membrane h[i_{11} \ldots i_{kn}] does not depend solely on the existence of h[i_{11} \ldots i_{k(n-1)_1}] and on the eventual application of (e)- or (f)-type rules in this membrane. If any of the upper level membranes containing (recursively) h[i_{11} \ldots i_{k(n-1)_1}] is divided at step \( n \) using a rule of type (f), then each of its sub-membranes is moved into only one of the two resulting membranes. Therefore, the existence of h[i_{11} \ldots i_{kn}] depends also on all indices i_{1n}, i_{2n}, \ldots, i_{kn} and on the behavior of all the upper-level membranes. We test this dependence recursively.

Function State

Parameters: h[i_{11} \ldots i_{kn}], n.


(i) If \( n = 0 \) then return the state of membrane h in the initial configuration and exit.

(ii) /* We check the existence of membrane h[i_{11} \ldots i_{kn}] w.r.t. a possible application of type (f) rules in upper level membranes. */
(a) X ← (\emptyset, 0), L ← false.
(b) If \( k = 0 \) (i.e. h is the skin membrane), go to step (e).
(c) g[i_{11} \ldots i_{kn}] ← Parent(h[i_{11} \ldots i_{kn}], X, L, α₃, α₄).
/* The function Parent simultaneously stores to X a state of the parent membrane, and also sets variables L, α₃, α₄, which indicate a possible polarization of membrane h. */
(d) If $X = nil$ then return nil and exit.

/* If the parent membrane $g[i_{11} \ldots i_{k(n-1)}]$ does not exist, neither does its child $h[i_{11} \ldots i_{kn}]$ exist. */

(e) $S \leftarrow \text{State}(h[i_{11} \ldots i_{k(n-1)}], n - 1)$.

(f) If $L = true$ then: if $S.p = +$ and $i_{kn} = 2$ or if $S.p = -$ and $i_{kn} = 1$ then return nil and exit.

/* The parent membrane $g[i_{11} \ldots i_{kn}]$ was divided with a type (f) rule so that its child $h[i_{11} \ldots i_{kn}]$ was moved to its copy different from that specified by $i_{kn}$. */

(iii) If $S = nil$ then return nil and exit.

/* If membrane $h[i_{11} \ldots i_{k(n-1)}]$ did not exist after $(n-1)$ steps, then after $n$ steps its successor $h[i_{11} \ldots i_{kn}]$ cannot exist. */

(iv) If $S = \text{dissolved}$ then:

if $i_{kn} = 2$ then return nil, else return dissolved, and exit.

(v) (a) $S'.M \leftarrow \emptyset$, $S'.p \leftarrow S.p$.

(b) If $L = true$ then:

- if $S.p = +$ then $S'.p \leftarrow \alpha_3$,
- if $S.p = -$ then $S'.p \leftarrow \alpha_4$.

/* If $L$ is true, the parent membrane $g$ was divided via a type-(f) rule which determines the polarization of its child $h[i_{11} \ldots i_{kn}]$. */

(vi) (a) $\text{Nplus} \leftarrow \emptyset$, $\text{Nminus} \leftarrow \emptyset$, $\text{Elementary} \leftarrow true$.

(b) $\text{Contribution_from_children}(h[i_{11} \ldots i_{kn}], S, S', \text{Elementary}, \text{Nplus}, \text{Nminus})$.

(vii) $X.M \leftarrow \emptyset$, $X'.M \leftarrow \emptyset$, $X.p \leftarrow 0$, $X'.p \leftarrow 0$.

(viii) If $h$ is not the skin membrane, then

$g[i_{11} \ldots i_{k(n-1)}] \leftarrow \text{Parent}(h[i_{11} \ldots i_{k(n-1)}], X, L, \alpha_3, \alpha_4)$.

(ix) /* We now simulate the evolution of membrane $h[i_{11} \ldots i_{k(n-1)}]$ at the $n$th step. */

(a) $\text{Try_rules_a}(h, S, S', X)$.

(b) $\text{Try_rules_b}(h, S, S', X)$; if any rule was applied, go to step (x).

(c) $\text{Try_rules_c}(h, S, S', X)$; if any rule was applied, go to step (x).

(d) $\text{Try_rules_d}(h, S, S', X)$; if any rule was applied, go to step (x).

(e) If $\text{Elementary}$ then:

- if $i_{kn} = 1$, then $\text{Try_rules_e}(h, S, S', X)$,
- if $i_{kn} = 2$, then $\text{Try_rules_e}(h, S, X, S')$.

(f) If not $\text{Elementary}$ then:

- if $i_{kn} = 1$, then $\text{Try_rules_f}(h, S, S', X, \text{Nplus}, \text{Nminus})$,
- if $i_{kn} = 2$, then $\text{Try_rules_f}(h, S, X, S', \text{Nplus}, \text{Nminus})$.

(x) If $i_{kn} = 2$ and neither a rule of type (e) nor (f) was applied, then $S' \leftarrow nil$.

/* If $i_{kn} = 2$, then membrane $h[i_{11} \ldots i_{kn}]$ could only be created by an application of an (e) or (f) type rule during the $n$th step. If such a rule was not applied, then $h[i_{11} \ldots i_{kn}]$ does not exist. */

(xi) If $S' \neq nil$ and $S' \neq \text{dissolved}$ then $S'.M \leftarrow S'.M \cup \text{S.M}$.

(xii) Return $S'$ and exit.

Procedure $\text{Try_rules_f}$ is implemented as follows:

Parameters: $h$, $S$, $S'$, $T$, $\text{Nplus}$, $\text{Nminus}$.

(f) For each rule $[h_0(h_1 \ldots h_i)_{h_{j+1}} \ldots [h_{j+1} \ldots [h_j]_{h_{j+1}} \ldots h_{j}]_{h_{j+1}}]^{a_2}_{h_0}$

$\rightarrow [h_0(h_1 \ldots h_i)_{h_{j+1}} \ldots [h_{j} h_{j+1}]^{\alpha_3}_{h_0} \ldots [h_{j}]_{h_{j+1}}^{\alpha_4}]_{h_0}$

such that $S.p = \alpha_2$, $\text{Nplus} = \{h_1, \ldots, h_i\}$ and $\text{Nminus} = \{h_{i+1}, \ldots, h_j\}$:

- $S'.p \leftarrow \alpha_5$,
- $T.p \leftarrow \alpha_6$,
- skip all other applicable rules.
Notice that during an application of a rule of type (f) only the polarization of membranes is changed but not their contents. The copying of sub-membranes contained in the dividing membrane was simulated solely by attaching digits 1 and 2 to their indices.

Again, observe that the recursive function \( \text{State} \) is defined correctly because each recursive call during the computation of \( \text{State}(h[i_{11} \ldots i_{kn}], n) \) is in one of the forms

\[
\begin{align*}
\text{State}(g[i_{11} \ldots i_{(k-1)n}], n), \\
\text{State}(g[i_{11} \ldots i_{k'(n-1)}], n-1), \quad 0 \leq k' \leq d,
\end{align*}
\]

where \( d \) is the depth of the initial membrane structure \( \mu \). By (7) and (8), each such call decreases the value of at least one of parameters \( n, k \geq 0 \). As moreover \( k \) is bounded from above, the resulting graph of recursive calls is acyclic and finite. Recursive calls in functions Parent and Contribution_from_children can be analyzed similarly, see (10)–(16) for details.

- **Space complexity of the simulation**

Consider an instance of a size \( s \) of a decision problem which is by assumption solved by a confluent P system \( \Pi = (V, H, \mu, w_1, \ldots, w_m, R) \) of a size \( s^{O(1)} \), a member of a semi-uniform family. A result of its computation can be calculated with the aid of the function \( \text{State} \). Let \( h_0 \) be the skin membrane of \( \Pi \). One can subsequently calculate \( \text{State}(h_0, n) \), until the object yes is expelled using the rule of type (c), or until the computation halts. Halting can be tested by computing \( \text{State}(h, n) \) 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 \( \text{State} \). Let

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

By the assumption, the values of \( m, d, \log p, q \) and \( \log o_0 \) are bounded from above by \( s^{O(1)} \) (which is the initial size of \( \Pi \)). In the rest of the proof we treat them as constants as they are fixed for a given \( \Pi \). Evaluating \( o_n \), if we considered only the rules of type (a), we would obtain \( o_n \leq o_0 p^n \). But the membranes can divide, too, 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 is

\[
o_n \leq o_0 m(p2^d)^n.
\]

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 space (in bits) necessary to store the contents of an arbitrary membrane after \( n \) steps is

\[
b_n \leq q\lceil \log o_n \rceil \leq q\lceil \log (o_0 m) \rceil + nq(\lceil \log p \rceil + d) = c_0 + c_1 n
\]

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

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

(i) a specification of membrane \( h[i_{11} \ldots i_{kn}] \) which requires \( kn + \lceil \log m \rceil \) bits, where \( m \) is the initial degree of the system \( \Pi \),

(ii) variables as \( S, S', X, X', Y \) 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,

(iii) some other variables such as \( L, Nplus \) and \( Nminus \) of a constant size \( c \) independent of \( n \) and \( k \).

We need not consider procedure parameters as all the structured parameters of the types described in paragraphs 1 and 2 above are passed by reference.
Denote the space complexity of the functions State, Parent and Contribution_from_children with the parameter $h[i_1 \ldots i_{kn}]$ by $S(n,k)$, $P(n,k)$ and $C(n,k)$, 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 corresponds to the following recurrences:

\begin{align}
S(0,k) &= b_0, \quad 0 \leq k \leq d, \\
S(n,0) &= C(n,0) + 4b_n + c, \quad n \geq 1, \\
S(n,k) &= \max \{ P(n,k), P(n-1,k), C(n,k), S(n-1,k) \} + 4b_n + kn + c, \quad n \geq 1, \quad 1 \leq k \leq d, \\
C(n,d) &= 0, \\
C(n,k) &\leq \max \{ C(n,k+1) + 3b_n + c, S(n-1,k+1) \} + n(k+1), \quad 0 \leq k < d, \quad n \geq 0, \\
P(n,0) &= S(n,0), \\
P(n,k) &\leq \max \{ P(n-1,k), S(n,k-1) \} + n(k-1) + c, \quad 1 \leq k \leq d, \quad n \geq 0.
\end{align}

By expanding (14) to a series for $k,k+1, \ldots ,d$ we obtain

\begin{equation}
C(n,k) \leq \max \{ S(n-1,i) \mid k < i \leq d \} + \mathcal{O}(d^2n + db_n)
\end{equation}

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

\begin{equation}
S(n) = \max \{ S(n,k) \mid 0 \leq k \leq d \}.
\end{equation}

Observe that one can omit $P(n-1,k)$ in (12) as obviously $P(n-1,k) \leq P(n,k)$. By (17) and (18) we can rewrite (12) in the form

\begin{equation}
S(n,k) \leq \max \{ P(n,k), S(n-1) + \mathcal{O}(d^2n + db_n) \} + \mathcal{O}(b_n + kn), \quad n \geq 1, \quad 1 \leq k \leq d.
\end{equation}

By substituting $P(n,i)$, $i = k,k-1, \ldots ,0$ with (16), and $S(n,i)$, $i = k-1,k-2, \ldots ,0$ with (19), we can expand (19) as follows:

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

The next-to-last step was done by substituting $P(n,0)$ and $S(n,0)$ with (15) and (11), respectively. In the last step we substituted $C(n,0)$ with (17) and (18). Therefore, the recurrence (10)–(12) can be rewritten with the aid of (18) as follows:

\begin{align*}
S(0) &= b_0, \\
S(n) &\leq S(n-1) + \mathcal{O}(d^2n + db_n).
\end{align*}

A solution to this recurrence is $S(n) = \mathcal{O}(d^2n^2 + ndb_n)$. Recall that $d = s^\mathcal{O}(1)$, where $s$ is the original instance size. By (9) we get

\begin{equation}
S(n) = (sn)^{\mathcal{O}(1)}.
\end{equation}

Finally, assuming that $\Pi$ is polynomial time-bounded, we have also $n = s^{\mathcal{O}(1)}$. After substituting to (20) one can conclude that the simulation is done in $\text{PSPACE}$.

Together with (5) we obtain the parallel computation thesis for uniform families of confluent P systems with active membranes:

**Corollary 3.** $\text{PMC}_{\text{AM}} = \text{PMC}_{\text{AM}}^S = \text{PSPACE}$. 

5. Concluding remarks

The results presented in this paper, as well as those in [1,16] and other cited sources, should be interpreted as a limit to the potential of natural computations, rather than a practical guide to the construction of a “wet computer” capturing \textit{PSPACE}. For instance, some operations used in P systems with active membranes, as the non-elementary membrane division, seem to have in practice very limited scalability, on one hand. On the other hand, we conjecture that this operation can be substituted by other means as complex membrane signals, tissue organization of membranes [9] etc. One should also note that most of the models of natural computing attacking NP- or \textit{PSPACE}-complete problems are based on trading space for time, which again implies limits of their scalability. However, if implemented “in bio,” they could still outperform silicon supercomputers due to their massive parallelism, minimal energy consumption, nanoscopic dimensions etc.

The proof technique we have used in Theorem 2 is applicable also to other variants of P systems with active membranes. For example, in [4] the non-elementary membrane division is controlled by rules of the form \([h a]_h^{\alpha_1} \rightarrow [h b]_h^{\alpha_2} [h c]_h^{\alpha_3}\). Eventual simulation of this variant would be analogous to the first part of our proof, and we claim Theorem 2 would remain valid. Another idea in [4] is the use of \textit{minimal parallelism}—only a non-deterministically chosen (possibly non-empty) subset of rules is applied in each membrane. As long as the P system remained confluent, the presented result would remain valid also in this case. Yet another possibility is a P system computing optimization problems. Such a P system would output a sequence of objects coding the optimal solution. One could also introduce cooperative rules (involving more than one object) or priorities among rules. The number of possible polarizations may be increased from three \((0, +, -)\) to a larger set, other features like promoters, inhibitors, impermeable membranes etc., may be used, without altering the proof structure.

Finally, we note that the characterization of power of non-confluent P systems with active 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. It is possible that non-confluent P systems with active membranes might capture in polynomial time the class \textit{NEXPTIME}.

Acknowledgments

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