Solving NP-complete Problems in Polynomial Time by Using a Natural Computing Model

Bogdan Aman(✉) and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Blvd. Carol I no. 8, 700505 Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Abstract. The first part of the paper is devoted to a polynomial solution of a well-known NP-complete problem (SAT problem) by using an unconventional computation model provided by P systems with active membranes (with neither polarization nor division rules). An important step of this semi-uniform solution is given by polynomial computing devices to build P systems that contain some exponential-size feature for which solving the SAT problem is easy. NP-complete problems are decision problems that can be solved in polynomial time on a nondeterministic Turing machine. Related to this step, in the second part we show how we can simulate polynomial space Turing machines by using a logarithmic space P system with active membranes, and employing a binary representation in order to encode the positions on the Turing machine tape.

Keywords: Natural computing · Membrane computing · Turing machines

1 Introduction

Membrane computing [15] is a branch of the natural computing inspired by the architecture and behaviour of living cells. Membrane systems (also called P systems) have been introduced by the computer scientist Gheorghe Păun, whose last name is the origin of the letter P in "P Systems". Membrane systems are characterized by three features: (i) a membrane structure consisting of a hierarchy of membranes (which are either disjoint or nested), with an unique top membrane called the *skin*; (ii) multisets of objects associated with membranes; (iii) rules for processing the objects and membranes. When membrane systems are seen as computing devices, two main research directions are usually considered: computational power in comparison with the classical notion of Turing computability (e.g., [2]), and efficiency in algorithmically solving NP-complete problems in polynomial time (e.g., [3]). Such efficient algorithms are obtained by trading space for time, with the space grown exponentially in a linear time by means of bio-inspired operations (e.g., membrane division). Thus, membrane systems define classes of computing devices which are both powerful and efficient.

V. Yakovyna et al. (Eds.): ICTERI 2015, CCIS 594, pp. 91–108, 2016.
DOI: 10.1007/978-3-319-30246-1_6

Related to the investigations of these research directions, there have been studied several applications of these systems; among them, modelling of various biological phenomena and the complexity and emergent properties of such systems presented in [7]. In [4] it is presented the detailed functioning of the sodium-potassium pump, while in [1] it is described and analyzed the immune system in the formal framework of P systems. Under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency of the system ways to exponentially grow the workspace during the computation, non-determinism, and so on). For instance, some pre-computed resources are used in [9].

We show that P systems with active membranes [13] can provide simple semi-uniform solutions to the SAT problem without using neither polarization nor division, but using exponential size pre-computed initial configurations (either alphabet or structure). An important observation is that we specify how our pre-computed initial configurations are constructed in a polynomial number of steps by additional well-defined P systems (P systems with replicated rewriting and P systems with active membranes and membrane creation, respectively).

The semi-uniform solutions rely on constructing the system and the solution in polynomial time in order to avoid solving the problem during the evolution of the system. In this context, the initial (exponential) configuration is constructed by another (polynomial) system, and the problem is solved by combining these two systems. In this way, we propose a polynomial solution that uses a polynomial P system for constructing the initial configuration (that is exponential). Related to this step, we show that P systems with active membranes provide an interesting simulation of polynomial space Turing machines by using only logarithmic space and a polynomial number of read-only input objects.

The rest of this paper is organized as follows: Sect. 2 contains some preliminary notions used in this paper. Section 3 provides simple semi-uniform solutions to the SAT problem, while Sect. 4 provides a simulation of polynomial space Turing machines by using only logarithmic space P systems with active membranes. Conclusion and references end the paper.

2 Preliminaries

We consider P systems with active membranes extended with an input alphabet, and such that the input objects cannot be created during the evolution [17]. The original definition also includes division rules, rules that are not needed here. The version used in this paper is similar to evolution-communication P systems used in [6] with additional read-only input objects and polarities.

Definition 1. *A* P system with active membranes and input objects *is a tuple*

$$\Pi = (\Gamma, \Delta, \Lambda, \mu; w_1, \ldots, w_d, R)$$

Where:

- $d \geq 1$ *is the* initial degree*;*
- Γ *is a finite non-empty alphabet of* objects*;*

- Δ *is an* input alphabet *of objects such that* $\Delta \cap \Gamma = \emptyset$*;*
- Λ *is a finite* set of labels *for membranes;*
- μ *is a* membrane structure *(i.e., a rooted unordered tree, usually represented by nested brackets) in which each membrane is labelled by an element of* Λ *in a one-to-one way, and possesses an attribute called* electrical charge*, which can be either neutral (0), positive (+) or negative (-);*
- $w_1, \ldots, w_d$ *are strings over* Γ*, describing the* initial multisets *of objects placed in a number of* d *membranes of* μ*; notice that* w_i *is assigned to membrane* i*;*
- R *is a finite* set of rules *over* $\Gamma \cup \Delta$*:*
  1. $[a \rightarrow w]_h^\alpha$ object evolution rules
     An object $a \in \Gamma$ *is rewritten into the multiset* w*, if* a *is placed inside a membrane labelled by* h *with charge* α*. An object* a *can be deleted by considering* w *the empty multiset* $\emptyset$*. Notice that these rules allow only to rewrite objects from* Γ*, but not from* Δ*.*
  2. $a[\ ]_h^\alpha \rightarrow [b]_h^\beta$ send-in communication rules
     An object a *is sent into a membrane labelled by* h *and with charge* α*, becoming* b*; also, the charge of* h *is changed to* β*. If* $b \in \Delta$*, then* $a = b$ *must hold.*
  3. $[a]_h^\alpha \rightarrow b[\ ]_h^\beta$ send-out communication rules
     An object a*, placed into a membrane labelled by* h *and having charge* α*, is sent out of membrane* h *and becomes* b*; simultaneously, the charge of* h *is changed to* β*. If* $b \in \Delta$*, then* $a = b$ *must hold.*
  4. $[a]_h^\alpha \rightarrow b$ dissolution rules
     An object a*, placed into a membrane labelled by* h *and having charge* α *dissolves membrane* h*and becomes* b*. All object contained in membrane* h *are released in the parent membrane of* h*.*

Each configuration $\mathcal{C}_i$ of a P system with active membranes and input objects is described by the current membrane structure, including the electrical charges, together with the multisets of objects located in the corresponding membranes. The initial configuration of such a system is denoted by $\mathcal{C}_0$. An evolution step from the current configuration $\mathcal{C}_i$ to a new configuration $\mathcal{C}_{i+1}$, denoted by $\mathcal{C}_i \Rightarrow \mathcal{C}_{i+1}$, is done according to the principles:

- Each object and membrane is involved in at most one communication rule per step.
- Each membrane could be involved in several object evolution rules that can be applied in parallel inside it.
- The application of rules is maximally parallel: the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules could be applied at the same time, a non-deterministic choice is performed; this implies that multiple configurations can be reached as the result of an evolution step.
- In each computation step, all the chosen rules are applied simultaneously.
- Any object sent out from the skin membrane cannot re-enter it.

A *halting evolution* of such a system Π is a finite sequence of configurations $\overrightarrow{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, such that $\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \ldots \Rightarrow \mathcal{C}_k$, and no rules can be applied any more in $\mathcal{C}_k$. A *non-halting* evolution $\overrightarrow{\mathcal{C}} = (\mathcal{C}_i \mid i \in \mathbb{N})$ consists of an infinite evolution $\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \ldots$, where the applicable rules are never exhausted.

Example 1. Addition is trivial; we consider n objects a and m objects b placed in a membrane 0 with charge +. The rule $[b \rightarrow a]_h^+$ says that an object b is transformed in one object a. Such a rule is applied in parallel as many times as possible. Consequently, all objects b are erased. The remaining number of objects a represents the addition $n + m$. More examples can be found in [5].

In order to solve decision problems (i.e., decide languages over an alphabet Σ), we use *families* of recognizer P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^*\}$ that respect the following conditions: (1) all evolutions halt; (2) two additional objects *yes* (successful evolution) and *no* (unsuccessful evolution) are used; (3) one of the objects *yes* and *no* appears in the halting configuration [16]. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^*$ by accepting or rejecting it. The mapping $x \vdash \Pi_x$ must be efficiently computable for each input length [12].

In this paper we use a logarithmic space uniformity condition [17].

Definition 2. *A family of P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^*\}$ is said to be $(\mathbf{L}, \mathbf{L})$-uniform if the mapping $x \vdash \Pi_x$ can be computed by two deterministic logarithmic space Turing machines F (for "family") and E (for "encoding") as follows:*

- *F computes the mapping $1^n \vdash \Pi_n$, where Π_n represents the membrane structure with some initial multisets and a specific input membrane, while n is the length of the input x.*
- *E computes the mapping $x \vdash w_x$, where w_x is a multiset encoding the specific input x.*
- *Finally, Π_x is Π_n with w_x added to the multiset placed inside its input membrane.*

In the following definition of space complexity adapted from [17], the input objects do not contribute to the size of the configuration of a P system. In this way, only the actual working space of the P system is measured, and P systems working in sublinear space may be analyzed.

Definition 3. *Given a configuration $\mathcal{C}$, the space size $|\mathcal{C}|$ is defined as the sum of the number of membranes in μ and the number of objects in Γ it contains. If $\overrightarrow{\mathcal{C}}$ is a halting evolution of Π, then $|\overrightarrow{\mathcal{C}}| = max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$ or, in the case of a non-halting evolution $\overrightarrow{\mathcal{C}}$, $|\overrightarrow{\mathcal{C}}| = sup\{|\mathcal{C}_i| \mid i \in \mathbb{N}\}$. The space required by Π itself is then $|\Pi| = sup\{|\overrightarrow{\mathcal{C}}| \mid \overrightarrow{\mathcal{C}}$ is an evolution of $\Pi\}$.*

Notice that $|\Pi| = \infty$ if Π has an evolution requiring infinite space or an infinite number of halting evolutions that can occur such that for each $k \in \mathbb{N}$ there exists at least one evolution requiring most than k steps.

3 Solving the SAT Problem with Active Membranes

At the beginning of 2005, Gh. Păun wrote:

> "*My favourite question (related to complexity aspects in P systems with active membranes and with electrical charges) is that about the number of polarizations. Can the polarizations be completely avoided? The feeling is that this is not possible - and such a result would be rather sound: passing from no polarization to two polarizations amounts to passing from non-efficiency to efficiency.*"

This conjecture (problem F in [14]) can be formally described in terms of membrane computing complexity classes as follows:

$$P = PMC_{\mathcal{AM}^0(+d,-n,+e,+c)}$$

where

- $PMC_{\mathcal{R}}$ indicates that the result holds for P systems with input membrane;
- $+d$ indicates that dissolution rules are permitted;
- $-n$ indicates that only division rules for elementary membranes are allowed;
- $+e$ indicates that evolution rules are permitted;
- $+c$ indicates that communication rules are permitted.

The SAT problem checks the satisfiability of a propositional logic formula in conjunctive normal form (CNF). Let $\{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables. A formula in CNF is of the form $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each C_i, $1 \leq i \leq m$ is a disjunction of the form $C_i = y_1 \vee y_2 \vee \cdots \vee y_r$ $(r \leq n)$, where each y_j is either a variable x_k or its negation $\neg x_k$.

We present some attempts to solve this conjecture by providing algorithms solving the SAT problem using P systems with active membranes with neither polarizations nor division, but using exponential pre-computed initial configurations constructed by additional P systems in polynomial time. As usually done in the membrane computing community, we construct effectively a system of membranes that solves the problem.

3.1 Solving SAT Problem by Using a Pre-computed Alphabet

In this section, we propose a polynomial time solution to the SAT problem using the polarizationless P systems with active membranes, without division, but with a pre-computed alphabet. For any instance of SAT we construct effectively a system of membranes that solves it. Formally, we prove the following result:

Theorem 1. *The SAT problem can be solved by a* **polarizationless** *P system with active membranes and* **without division**, *but with an exponential alphabet pre-computed in linear time with respect to the number of variables and the number of clauses, i.e.,*

$$P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\alpha))}.$$

where $pre(\alpha)$ *indicates that a pre-computed alphabet is permitted.*

Proof. Let us consider a propositional formula

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

where each $C_i, 1 \leq i \leq m$ is a disjunction of the form

$$C_i = y_1 \vee y_2 \vee \cdots \vee y_r (r \leq n),$$

where each y_j is either a variable x_k or its negation $\neg x_k$.

We construct a P system with active membranes able to check the satisfiability of φ. The P system is given by $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, where:

- $\Gamma = \{z_i \mid 0 \leq i \leq max\{m, n\}\} \cup$
  $\cup \{s_i \mid i = t_1 \ldots t_n, t_j \in \{0, 1\} \text{ and } 1 \leq j \leq n\} \cup \{yes, no\}$.
  The alphabet $\{s_i \mid i = t_1 \ldots t_n, t_j \in \{0, 1\} \text{ and } 1 \leq j \leq n\}$ to be placed inside the input membrane 0 can be generated, starting from an object s, using the rules:
  - $s \rightarrow s_0 s_1$;
  - $s_i \rightarrow s_{i0} s_{i1}$, for $i = t_1 \ldots t_k$ where $t_j \in \{0, 1\}$ and $1 \leq j \leq k < n$.

  Thus all the possible assignments for the n variable $\{x_1, x_2, \ldots, x_n\}$ are created. The rules are applied until the length k of i in the second rule equals n. For example, s_{100} over $\{x_1, x_2, x_3\}$ represents the assignment $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$ (1 stands for $true$, while 0 stands for $false$). The input alphabet can be computed in linear (polynomial) time by using an additional device, for instance P systems with replicated rewriting [8].
- $\Lambda = \{0, c_1, \ldots, c_m, h\}$, with $c_i = z_1 \ldots z_n$, $1 \leq i \leq m$ where
  - $z_j = 1$ if x_j appears in C_i;
  - $z_j = 0$ if $\neg x_j$ appears in C_i;
  - $z_j = \star$ if neither x_j nor $\neg x_j$ appear in C_i.

  For example $c_1 = 1 \star 0$ over the set of variables $\{x_1, x_2, x_3\}$ represents the disjunction $c_1 = x_1 \vee \neg x_3$.
- $\mu = [[[\ldots[[[\,]_0]_{c_1}]_{c_2} \ldots]_{c_{m-1}}]_{c_m}]_h$.
- $w_0 = z_0$.
- $w_i = \lambda$, for all $i \in \Lambda \backslash \{0\}$.
- The set R contains the following rules:
  1. $[z_0]_0 \rightarrow z_0$
     After the input is placed inside membrane 0, membrane 0 is dissolved, and its content is released in the upper membrane labelled with c_1.
  2. $[s_i]_{c_j} \rightarrow s_i[\,]_{c_j}$
     if i and j have at least one position with the same value (either 0 or 1);
  3. $[s_i]_{c_m} \rightarrow yes$
     if i and m have at least one position with the same value (either 0 or 1).
     An assignment s_i is sent out of a membrane c_m if there is at least one position in i and j that is equal, namely an assignment to a variable x_k such that it makes C_j true. Once an object yes is generated, another object yes cannot be created because membrane c_m was dissolved and the rule $[s_i]_{c_m} \rightarrow yes$ cannot be applied. For example, if $c_1 = 1 \star 0$ and s_{101}

(as described above), then this means that s_{101} satisfies the clause coded by $c_1 = 1 \star 0$ since both have 1 on their first position, and this is enough to make true a disjunction.

4. $[z_0 \rightarrow z_1]_{c_1}$
5. $[z_i]_{c_i} \rightarrow [\ ]_{c_i} z_{i+1}$, for $1 \leq i \leq m-1$
6. $[z_m]_{c_m} \rightarrow no$
   The object z_0 waits a step after membrane 0 is dissolved in order to allow the other objects s_i to go through the c_j membranes. The object z_i then is communicated through the c_j membranes. Once z_m reached the membrane c_m, if membrane c_m still exists (i.e., the rule $[s_i]_{c_m} \rightarrow yes$ was not applied), then the answer *no* is generated. Once an object *yes* or *no* is generated, other objects *yes* or *no* cannot be created because membrane c_m was dissolved, and neither rule $[s_i]_{c_m} \rightarrow yes$ nor $[z_m]_{c_m} \rightarrow no$ can be applied.
7. $[yes]_h \rightarrow yes[\ ]_h$
8. $[no]_h \rightarrow no[\ ]_h$
   The answer *yes* or *no* regarding the satisfiability is sent out of the skin.

3.2 Solving SAT Problem Using a Pre-computed Initial Structure

In this section, we propose a polynomial time solution to the SAT problem using the polarizationless P systems with active membranes and without division, but with a pre-computed structure. For any instance of SAT we construct effectively a system of membranes that solves it. The fact that each membrane can be subject to at most one communication rule per step is needed when generating all possible assignments to be verified. Formally, we prove the following result:

Theorem 2. *The SAT problem can be solved by a* **polarizationless** *P system with active membranes and* **without division**, *but with an initial exponential structure pre-computed in linear time with respect to the number of variables and the number of clauses, i.e.,*

$$P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\mu))}.$$

where $pre(\mu)$ indicates that a pre-computed structure is permitted.

Proof. Let us consider a propositional formula

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

where each $C_i, 1 \leq i \leq m$ is a disjunction of the form

$$C_i = y_1 \vee y_2 \vee \cdots \vee y_r (r \leq n),$$

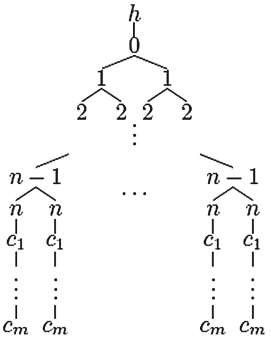
where each y_j is either a variable x_k or its negation $\neg x_k$.

We construct a P system with active membranes able to check the satisfiability of φ. The P system is given by $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, where:

– $\Gamma = \{a_i, t_i, t'_i, f_i, f'_i \mid 1 \leq i \leq n\} \cup \{z_i \mid 0 \leq i \leq 4 \times n + 2 \times m\} \cup \{yes, no\}$.

- $\Lambda = \{0, \ldots, n, c_1, \ldots, c_m, h\}$, $1 \leq i \leq m$.
- $\mu = [[[[\ldots]_2[\ldots]_2]_1[[\ldots]_2[\ldots]_2]_1]_0]_h$, where
  - each membrane i contains two membranes $i+1$ for $0 \leq i \leq n-1$;
  - each membrane n contains a membrane structure $[[\ldots[\ ]_{c_m}\ldots]_{c_1}]_{c_0}$;
  - membrane 0 is the input membrane.

  Graphically, the membrane structure μ can be represented as a tree:



  This membrane structure can be generate in linear (polynomial) time with respect to the number of variables and the number of clauses. This is done by using an additional device that starts from a membrane structure $[[\ ]_0]_h$, with object 0 placed inside membrane 0 and rules of the form:
  - $[i \to (i+1)'\ (i+1)']_i$, for $0 \leq i \leq n-1$
  - $i' \to [i]_i$, for $1 \leq i \leq n$
  - $n \to [c_2]_{c_1}$
  - $c_k \to [c_{k+1}]_{c_k}$, for $2 \leq k \leq m-1$
  - $c_m \to [\ ]_{c_m}$.
- $w_0 = a_1 z_0$.
- $w_i = \lambda$, for all $i \in \Lambda \backslash \{0\}$.
- The set R contains the following rules:
  1. $[z_i \to z_{i+1}]_0$, for all $0 \leq i < 4 \times n + 2 \times m$
     These rules count the time needed for producing the truth assignments for the n variables inside the membranes labelled by n ($3 \times n$ steps), then to dissolve the membranes labelled by c_j, $1 \leq j \leq m$ ($2 \times m$ steps), and for an y object to reach the membrane labelled by 0 (n steps).
  2. $[a_i \to t_i f_i]_{i-1}$, for $1 \leq i \leq n$
  3. $t_i[\ ]_i \to [t_i]_i$, for $1 \leq i \leq n$
  4. $f_i[\ ]_i \to [f_i]_i$, for $1 \leq i \leq n$
  5. $[t_i \to t_i' t_i' a_{i+1}]_i$, for $1 \leq i \leq n-1$
  6. $t_i'[\ ]_k \to [t_i]_k$, for $i+1 \leq k \leq n$
  7. $[t_i \to t_i' t_i']_k$, for $i+1 \leq k \leq n-1$
  8. $[f_i \to f_i' f_i' a_{i+1}]_i$, for $1 \leq i \leq n-1$
  9. $f_i'[\ ]_k \to [f_i]_k$, for $i+1 \leq k \leq n$
  10. $[f_i \to f_i' f_i']_k$, for $i+1 \leq k \leq n-1$

      In membranes n we create all possible assignments for the n variable $\{x_1, x_2, \ldots, x_n\}$. It starts from an object a_1 placed initially in membrane

labelled by 0. Each a_i is used to create t_i and f_i that are then send in one of the two membranes labelled by i placed in membrane $i-1$. In fact each membrane i receives either t_i or f_i, and this is possible because a membrane can be involved in only one communication rule of an evolution step. After an object t_i or f_i reaches a membrane i, it generates two new copies of it to be sent inside membranes $i+1$ together with an object a_{i+1} that is used then to construct the assignments over variable x_{i+1}.

11. $t_i[\ ]_{c_j} \to [t_i]_{c_j}$, if x_i appears in C_j
12. $[t_i]_{c_j} \to t_i$, for $1 \leq i \leq n$, $1 \leq j < m$
13. $[t_i]_{c_m} \to y$, for $1 \leq i \leq n$
14. $f_i[\ ]_{c_j} \to [t_i]_{c_j}$, if $\neg x_i$ appears in C_j
15. $[f_i]_{c_j} \to f_i$, for $1 \leq i \leq n$, $1 \leq j \leq m$
16. $[f_i]_{c_m} \to y$, for $1 \leq i \leq n$.
    An assignment t_i (f_i) is sent into a membrane c_j if there is an assignment to a variable x_k ($\neg x_k$) such that it makes C_j true. Once all membranes labelled by c_i are dissolved inside a membrane labelled by n, an object y is generated.
17. $[y]_k \to [\ ]_k y$, for $k \in \Lambda \backslash \{0, h\}$
18. $[y]_0 \to yes$
19. $[z_{4\times n+2\times m}]_0 \to no$.
    The object z_0 waits for $4 \times n + 2 \times m$ steps in order to allow dissolving the membrane labelled by 0 if this still exists (i.e., the rule $[y]_0 \to yes$ was not applied), then the answer *no* is generated. Once an object *yes* or *no* is generated, other objects *yes* or *no* cannot be created because membrane c_m was dissolved, and neither rule $[y]_0 \to yes$ nor $[z_{4\times n+2\times m}]_0 \to no$ can be applied.
20. $[yes]_h \to yes[\ ]_h$
21. $[no]_h \to no[\ ]_h$.
    The answer *yes* or *no* regarding the satisfiability is sent out of the skin.

Example 2. We illustrate this algorithm and the evolution of a system Π constructed for the propositional formula $\psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$.

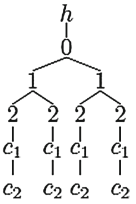
Thus, m = n = 2. The initial configuration of the systems, constructed by an additional device that starts from a membrane structure $[[\ ]_0]_h$, with object 0 placed inside membrane 0 and rules of the form:

- $[0 \to 1'\ 1']_0$ and $[1 \to 2'\ 2']_1$
- $1' \to [1]_1$ and $2' \to [2]_2$
- $2 \to [c_2]_{c_1}$ and $c_2 \to [\ ]_{c_2}$.

The obtained structure is

$$[[[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1[[[[\ ]_{c_2}]_{c_1}]_2[[[\ ]_{c_2}]_{c_1}]_2]_1 a_1 z_0]_0]_h$$

Graphically, the membrane structure μ can be represented as a tree:



Using the set R of rules $1 \div 21$, the computation proceeds as follows:

$$
\begin{aligned}
& [[[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2]_1[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2]_1 a_1 z_0]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2]_1[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2]_1 t_1 f_1 z_1]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2 t_1]_1[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2 f_1]_1 z_2]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2 t_1' t_1' a_2]_1[[[[\,]_{c_2}]_{c_1}]_2[[[\,]_{c_2}]_{c_1}]_2 f_1' f_1' a_2]_1 z_3]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2}]_{c_1} t_1]_2[[[\,]_{c_2}]_{c_1} t_1]_2 t_2 f_2]_1[[[[\,]_{c_2}]_{c_1} f_1]_2[[[\,]_{c_2}]_{c_1} f_1]_2 t_2 f_2]_1 z_4]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2}]_{c_1} t_1 t_2]_2[[[\,]_{c_2}]_{c_1} t_1 f_2]_2]_1[[[[\,]_{c_2}]_{c_1} f_1 t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_5]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1]_{c_1} t_2]_2[[[\,]_{c_2} t_1]_{c_1} f_2]_2]_1[[[[\,]_{c_2} t_2]_{c_1} f_1]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_6]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1 t_2]_2[[\,]_{c_2} t_1 f_2]_2]_1[[[\,]_{c_2} t_2 f_1]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_7]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1 t_2]_2[[f_2]_{c_2} t_1]_2]_1[[[f_1]_{c_2} t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_8]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1 t_2]_2[y t_1]_2]_1[[y t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 z_9]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1 t_2]_2[t_1]_2 y]_1[[t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2 y]_1 z_{10}]_0]_h \\
\Rightarrow\ & [[[[[\,]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 yy z_{11}]_0]_h \\
\Rightarrow\ & [[[[\,]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 y z_{12} yes]_h \\
\Rightarrow\ & [[[[\,]_{c_2} t_1 t_2]_2[t_1]_2]_1[[t_2]_2[[[\,]_{c_2}]_{c_1} f_1 f_2]_2]_1 y z_{12}]_h yes
\end{aligned}
$$

It can be noticed that even the object z has now the subscript $4 \times n + 2 \times m = 4 \times 2 + 2 \times 2 = 12$, it cannot generate a *no* object because membrane labelled by 0 was already dissolved by an y object in the previous step. Also, even another y object reached the membrane labelled by 0, it cannot generate an *yes* object because membrane labelled by 0 was already dissolved by another y object in a previous step.

4 Natural Computing Modelling of the Polynomial Space Turing Machines

The semi-uniform solutions rely on constructing the system and the solution in polynomial time in order to avoid solving the problem during the evolution of the system. In this context, the initial (exponential) configuration is constructed by another (polynomial) system, and the problem is solved by combining these two systems. In this way, we propose a polynomial solution that uses a polynomial P system for constructing the initial configuration (that is exponential). Related to this step, we show that P systems with active membranes provide an interesting

simulation of polynomial space Turing machines by using only logarithmic space and a polynomial number of read-only input objects.

4.1 A Membrane Structure for Simulation

Let M be a single-tape deterministic Turing machine working in polynomial space $s(n)$. Let Q be the set of states of M, including the initial state s; we denote by $\Sigma' = \Sigma \cup \{\sqcup\}$ the tape alphabet which includes the blank symbol $\sqcup \notin \Sigma$. A computation step is performed by using $\delta : Q \times \Sigma' \to Q \times \Sigma' \times \{-1, 0, 1\}$, a (partial) transition function of M which we assume to be undefined on (q, σ) if and only if q is a final state. We describe a uniform family of P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^*\}$ simulating M in logarithmic space.

Let $x \in \Sigma^n$ be an input string, and let $m = \lceil log\ s(n) \rceil$ be the minimum number of bits needed in order to write the tape cell indices $0, \ldots, s(n)$-1 in binary notation. The P system Π_n associated with the input length n and computed as $F(1^n)$ has a membrane structure consisting of $|\Sigma'| \cdot (m + 1) + 2$ membranes. The membrane structure contains:

- a skin membrane h;
- an inner membrane c (the input membrane) used to identify the symbol needed to compute the δ function;
- for each symbol $\sigma \in \Sigma'$ of M, the following set of membranes, linearly nested inside c and listed inward:
  - a membrane σ for each symbol σ of the tape alphabet Σ' of M;
  - for each $j \in \{0, \ldots, (m-1)\}$, a membrane labelled by j.

This labelling is used in order to simplify the notations. To respect the one-to-one labelling from Definition 1, the membrane j can be labelled j_σ. Thus in all rules using membranes j, the σ symbol is implicitly considered. Furthermore, the object z_0 is located inside the skin membrane h.

The encoding of x, computed as $E(x)$, consists of a set of objects describing the tape of M in its initial configuration on input x. These objects are the symbols of x subscripted by their position $bin(0), \ldots, bin(n-1)$ (where $bin(i)$ is the binary representation of i on m positions) in x, together with the $s(n) - n$ blank objects subscripted by their position $bin(n), \ldots, bin(s(n)-1)$. The binary representation, together with the polarities of the membranes, is essential when the membrane system has to identify the symbol needed to simulate the δ function (e.g., rule (13)). The multiset $E(x)$ is placed inside the input membrane c. Figure 1 depicts an example.

During the first evolution steps of Π_x, each input object σ_i is moved from the input membrane c to the innermost membrane $(m-1)$ of the corresponding membrane σ by means of the following communication rules:

$$\sigma_i[\ ]^0_\sigma \to [\sigma_i]^0_\sigma \quad \text{for} \quad \sigma \in \Sigma',\ bin(0) \le i < bin(s(n)) \tag{1}$$

$$\sigma_i[\ ]^0_j \to [\sigma_i]^0_j \quad \text{for} \quad \sigma \in \Sigma',\ bin(0) \le i < bin(s(n)),\ 0 \le j < m \tag{2}$$

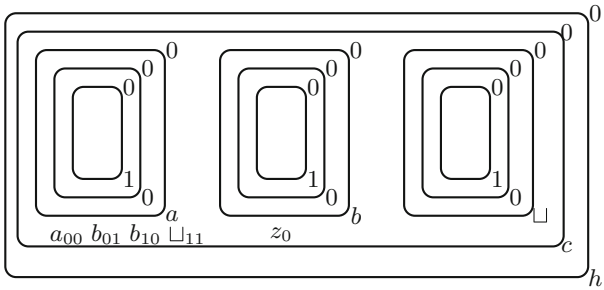


Fig. 1. Initial configuration of the P system Π_3 with tape alphabet $\Sigma' = \{a, b, \sqcup\}$, working in space $s(n) = n + 1 = 4$ on the input *abb*.

Since only one communication rule per membrane can be applied during each evolution step of Π_x, all $s(n)$ input objects pass through m membranes, in order to reach the innermost membranes $(m-1)$, in at most $l = s(n) + m$ evolution steps. In the meantime, the subscript of object z_0 is incremented up to $max\{0, l-3\}$ before object z_{l-3} exits and enters membrane c changing the membrane charge from 0 to +:

$$[z_t \to z_{t+1}]_c^0 \qquad \text{for} \quad 0 \le t < l-3 \tag{3}$$

$$[z_{l-3}]_c^0 \to z_{l-3}[\,]_c^0 \tag{4}$$

$$z_{l-3}[\,]_c^0 \to [z_{l-3}]_c^+ \tag{5}$$

The object z_{l-3} is rewritten to a multiset of objects containing an object z' (used in rule (10)) and $|\Sigma'|$ objects z_+ (used in rules (7))

$$[z_{l-3} \to z' \underbrace{z_+ \cdots z_+}_{|\Sigma'|\text{copies}}\,]_c^+ \tag{6}$$

The objects z_+ are used to change the charges from 0 to + for all membranes $\sigma \in \Sigma'$ using parallel communication rules, and then are deleted:

$$z_+[\,]_\sigma^0 \to [\#]_\sigma^+ \quad \text{for} \quad \sigma \in \Sigma' \tag{7}$$

$$[\# \to \emptyset]_\sigma^+ \qquad \text{for} \quad \sigma \in \Sigma' \tag{8}$$

In the meantime, the object z' is rewritten into z'' (in parallel with rule (7)), and then, in parallel with rule (8), into s_{00} (where s is the initial state of M):

$$[z' \to z'']_c^+ \tag{9}$$

$$[z'' \to s_{00}]_c^+ \tag{10}$$

The configuration reached by Π_x encodes the initial configuration of M (Fig. 2):

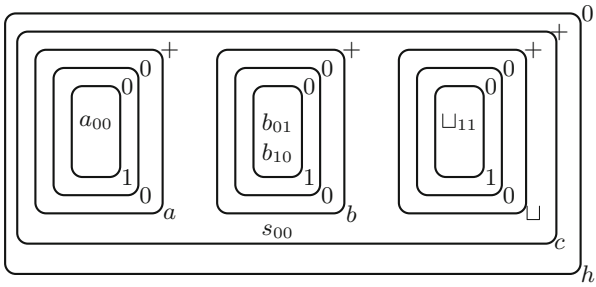


Fig. 2. Configuration of Π_x (from Fig. 1) encoding the initial configuration of M on input $x = abb$ and using $s(|x|) = 4$ tape cells.

An arbitrary configuration of M on input x is encoded by a configuration of Π_x as it is described in Fig. 3:

- membrane c contains the state-object q_i, where q is the current state of M and $i \in \{bin(0), \ldots, bin(s(n) - 1)\}$ is the current position of the tape head;
- membranes $(m - 1)$ contain all input objects;
- all other membranes are empty;
- all membranes are neutrally charged, except those labelled by $\sigma \in \Sigma'$ and by c which all are positively charged.

We employ this encoding because the input objects must be all located in the input membrane in the initial configuration of Π_x (hence they must encode both symbol and position on the tape), and they can never be rewritten.

4.2 Simulating Polynomial Space Turing Machines

Starting from a configuration of the single-tape deterministic Turing machine M, the simulation of a computation step of M by the membrane system Π_x is directed by the state-object q_i. As stated above, q_i encodes the current state

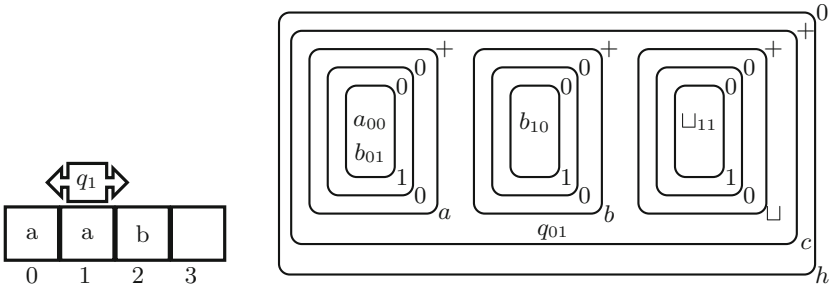


Fig. 3. A configuration of M (from Fig. 1) and the corresponding configuration of Π_x simulating it. The presence of b_{01} inside membrane 1 of a indicates that tape cell 1 of M contains the symbol a.

of M and the position of the head on the tape (in binary format). To simulate the transition function δ of the Turing machine M in state q, it is necessary to identify the actual symbol occurring at tape position i. In order to identify this σ_i object from one of the $(m-1)$ membranes, the object q_i is rewritten into $|\Sigma'|$ copies of q'_i, one for each membrane $\sigma \in \Sigma'$:

$$[q_i \to \underbrace{q'_i \cdots q'_i}_{|\Sigma'| \text{copies}}]_c^+ \quad \text{for } q \in Q,\ bin(0) \leq i < bin(s(n)) \tag{11}$$

The objects q'_i first enter the symbol-membranes in parallel, without changing the charges:

$$q'_i[\ ]_\sigma^+ \to [q'_i]_\sigma^+ \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \tag{12}$$

The object q'_i traverses the membranes $0, \ldots, (m-1)$ while changing their charges such that they represent the bits of i from the least to the most significant one, where a positive charge is interpreted as 1 and a negative charge as 0. For instance, the charges of $[[[\ ]_2^-]_1^-]_0^+$ encode the binary number 001 (that is, decimal 1). By the j-th bit of a binary number is understood the bit from the j-th position when the number is read from right to left (e.g., the 0-th bit of the binary number 001 is 1). The changes of charges are accomplished by the rules:

$$q'_i[\ ]_j^0 \to [q'_i]_j^\alpha \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)),\ 0 \leq j < m, \tag{13}$$

where α is $-$ if the j-th bit of i is 0, and α is $+$ if the j-th bit of i is 1.

The membranes j, where $0 \leq j < m$, behave now as "filters" for the input objects σ_k occurring in membrane $(m-1)$: these objects are sent out from each membrane j if and only if the j-th bit of k corresponds to the charge of j.

$$[\sigma_k]_j^\alpha \to [\ ]_j^\alpha \sigma_k \quad \text{for } \sigma \in \Sigma',\ bin(0) \leq k < bin(s(n)),\ 0 < j < m, \tag{14}$$

where α is $-$ if the j-th bit of k is 0, and α is $+$ if j-th bit of k is 1.

If an object σ_k reaches membrane 0, it is sent outside if the 0-th bit of k corresponds to the charge of membrane 0. In order to signal that it is the symbol occurring at location i of the tape, the charge of the corresponding membrane 0 is changed (either from $+$ to $-$ or from $-$ to $+$). By applying the rules (15) to (17), exactly one object σ_k, with $k = i$, will exit through membrane c:

$$\begin{array}{l} [\sigma_k]_0^+ \to [\ ]_0^- \sigma_k \quad \text{for } \sigma \in \Sigma',\ bin(0) \leq k < bin(s(n)) \\ [\sigma_k]_0^- \to [\ ]_0^+ \sigma_k \quad \text{for } \sigma \in \Sigma',\ bin(0) \leq k < bin(s(n)), \end{array} \tag{15}$$

where α is $-$ if the j-th bit of k is 0, and α is $+$ if the j-th bit of k is 1;

$$[\sigma_k]_\tau^+ \to \sigma_k[\ ]_\tau^- \quad \text{for } \sigma, \tau \in \Sigma',\ bin(0) \leq k < bin(s(n)). \tag{16}$$

After an σ_i exits from membrane c it gets blocked inside membrane h, by the new charge of membrane c, until it is allowed to move to its new location according to function δ of the Turing machine M. Thus, if another object τ_j reached

membrane σ due to the new charge of membrane 0 established by rule (15), τ_j is contained in membrane σ until reintroduced in a membrane (m-1) using rule (2).

$$[\sigma_k]_c^+ \rightarrow \sigma_k[\,]_c^- \quad \text{for } \sigma \in \Sigma', bin(0) \leq k < bin(s(n)) \tag{17}$$

Since there are $s(n)$ input objects, and each of them must traverse at most $(m+1)$ membranes, the object σ_i reaches the skin membrane h after at most $l+1$ steps, where l is as defined in Sect. 4.1 before rule (3). While the input objects are "filtered out", the state-object q_i' "waits" for l steps using the rules:

$$[q_i' \rightarrow q_{i,1}'']_{m-1}^{\alpha} \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)), \alpha \in \{-,+\} \tag{18}$$

$$[q_{i,t}'' \rightarrow q_{i,t+1}'']_{m-1}^{\alpha} \quad \begin{array}{l}\sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \\ \alpha \in \{-,+\},\ 1 \leq t \leq l\end{array} \tag{19}$$

$$[q_{i,l+1}'' \rightarrow q_i'']_{m-1}^{\alpha} \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)), \alpha \in \{-,+\} \tag{20}$$

In order to reach membrane c, the objects q_i'' are sent out through membranes j $(0 < j \leq m-1)$ using rule (21), through membrane 0 by rules (22) and (24), and through membranes $\sigma \in \Sigma'$ by rule (23). While passing through all these membranes, the charges are changed to neutral. This allows the input objects to move back to the innermost membrane $(m-1)$ by using rules of type (2).

$$[q_i'']_j^{\alpha} \rightarrow [\,]_j^0 q_i'' \quad \begin{array}{c}\text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \\ 0 < j \leq m-1,\ \alpha \in \{-,+\}\end{array} \tag{21}$$

When q_i'' reaches the membranes 0, only one has the charge different from the 0-th bit of i, thus allowing q_i'' to identify the symbol in tape location i of M:

$$[q_i'']_0^{\alpha} \rightarrow [\,]_0^0 q_i''' \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)), \tag{22}$$

where α is $-$ if the 0-th bit of i is 1, and α is $+$ if the 0-th bit of i is 0.

$$[q_i''']_{\sigma}^- \rightarrow [\,]_{\sigma}^0 q_{i,\sigma,1} \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \tag{23}$$

The other copies of q_i'' are sent out as objects # through membrane 0, and then deleted by rules of type (8):

$$[q_i'']_0^{\alpha} \rightarrow [\,]_0^0 \# \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)), \tag{24}$$

where α is $-$ if the 0-th bit of i is 0, and α is $+$ if the 0-th bit of i is 1.

The state-object $q_{i,\sigma,1}$ waits in membrane c for l steps, l representing an upper bound of the number of steps needed for all the input objects to reach the innermost membranes:

$$[q_{i,\sigma,t} \rightarrow q_{i,\sigma,t+1}]_c^- \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)),\ 1 \leq t < l \tag{25}$$

$$q_{i,\sigma,l}[\,]_{\sigma}^0 \rightarrow [q_{i,\sigma,l}]_{\sigma}^+ \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \tag{26}$$

$$[q_{i,\sigma,l}]_{\sigma}^+ \rightarrow q_{i,\sigma}'[\,]_{\sigma}^+ \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \leq i < bin(s(n)) \tag{27}$$

The state-object $q'_{i,\sigma}$ now contains all the information needed to compute the transition function δ of the Turing machine M. Suppose $\delta(q,\sigma) = (r,v,d)$ for some $d \in \{-1,0,+1\}$. Then $q'_{i,\sigma}$ sets the charge of membrane v to $-$ and waits for $m+1$ steps, thus allowing σ_i to move to membrane $(m-1)$ of v by using the rules (31), (32) and (2):

$$q'_{i,\sigma}[\,]^+_v \to [q'_{i,\sigma}]^+_v \quad \text{for } \sigma, v \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{28}$$
$$[q'_{i,\sigma}]^+_v \to q'_{i,\sigma,1}[\,]^-_v \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{29}$$
$$[q'_{i,\sigma,1}]^-_c \to q'_{i,\sigma,1}[\,]^0_c \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{30}$$
$$\sigma_i[\,]^0_c \to [\sigma_i]^0_c \quad \text{for } \sigma \in \Sigma',\ bin(0) \le i < bin(s(n)) \tag{31}$$
$$\sigma_i[\,]^-_v \to [\sigma_i]^-_v \quad \text{for } \sigma, v \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{32}$$
$$[q'_{i,\sigma,t} \to q'_{i,\sigma,t+1}]^0_h \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) 1 \le t \le m \tag{33}$$

The object $q'_{i,\sigma,m+1}$ is used to change the charges of membranes c and v to $+$, thus preparing the system for the next step of the simulation:

$$q'_{i,\sigma,m+1}[\,]^0_c \to [q'_{i,\sigma,m+1}]^+_c \quad \text{for } \sigma \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{34}$$
$$q'_{i,\sigma,m+1}[\,]^-_v \to [q'_{i,\sigma,m+1}]^-_v \quad \text{for } \sigma, v \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{35}$$
$$[q'_{i,\sigma,m+1}]^-_v \to q''_{i,\sigma}[\,]^+_v \quad \text{for } \sigma, v \in \Sigma',\ q \in Q,\ bin(0) \le i < bin(s(n)) \tag{36}$$

Finally, the state-object $q''_{i,\sigma}$ is rewritten to reflect the change of state and head position, thus producing a configuration of Π_x corresponding to the new configuration of M, as described in Sect. 4.1:

$$[q''_{i,\sigma} \to r_{i+d}]^+_c \quad \text{for } bin(0) \le i < bin(s(n)) \tag{37}$$

The P system Π_x is now ready to simulate the next step of M. If $q \in Q$ is a final state of M, we assume that $\delta(q,\sigma)$ is undefined for all $\sigma \in \Sigma'$; thus we introduce the following rules which halt the P system with the same result (acceptance or rejection) as M:

$$[q_i]^+_c \to [\,]^+_c yes \quad \text{for } bin(0) \le i < bin(s(n)),\ \text{if } q \text{ is an accepting state} \tag{38}$$
$$[yes]^0_h \to [\,]^0_h yes \quad \text{for } bin(0) \le i < bin(s(n)),\ \text{if } q \text{ is an accepting state} \tag{39}$$
$$[q_i]^+_c \to [\,]^+_c no \quad \text{for } bin(0) \le i < bin(s(n)),\ \text{if } q \text{ is a rejecting state} \tag{40}$$
$$[no]^0_h \to [\,]^0_h no \quad \text{for } bin(0) \le i < bin(s(n)),\ \text{if } q \text{ is a rejecting state} \tag{41}$$

The simulation directly leads to the following result.

Theorem 3. *Let M be a single-tape deterministic Turing machine working in polynomial space $s(n)$ and time $t(n)$. Then there exists an $(\mathbf{L},\mathbf{L})$-uniform family $\mathbf{\Pi}$ of P systems Π_x with active membranes using object evolution and communication rules that simulates M in space $O(\log n)$ and time $O(t(n)s(n))$.*

Proof. For each $x \in \Sigma^n$, the P system Π_x can be built from 1^n and x in logarithmic space as it is described in Definition 2; thus, the family $\mathbf{\Pi}$ is $(\mathbf{L},\mathbf{L})$-uniform.

Each P system Π_x uses only a logarithmic number of membranes and a constant number of objects per configuration; thus, Π_x works in space $O(log\ n)$. Simulating one of the $t(n)$ steps of M requires $O(s(n))$ time, an upper bound to the subscripts of objects used to introduce delays during the simulation; thus, the total time is $O(t(n)s(n))$.

5 Conclusion

We proved $P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\alpha))}$ and $P = PMC_{\mathcal{AM}^0(+d,+e,+c,pre(\mu))}$ by providing two algorithms for solving the SAT problem using **polarizationless** P system with active membranes and **without division**. For the former equality, the provided algorithm is using an exponential alphabet pre-computed in linear time by a P system with replicated rewriting, while the later one is using an initial exponential structure pre-computed in linear time with respect to the number of variables and clauses by P systems with membrane creation.

In this paper we also provided a simulation of the polynomial space Turing machines by using logarithmic space P systems with active membranes and binary representations for the positions on the tape. A similar approach is presented in [11]. There are important differences in terms of technical details and efficient representation; in comparison to [11], we improve the simulation by reducing the number of membranes (by $|\Sigma'| - 1$) and the number of rules (by $|\Sigma'|\cdot|Q|\cdot s(n)\cdot(5-|\Sigma'|)+|\Sigma'|\cdot|\Sigma'|s(n)\cdot(2\cdot m+1)+|Q|\cdot s(n)-|\Sigma'|\cdot s(n)\cdot(m+3)$). In particular, for the running example, the number of rules is reduced by $14\cdot|Q|+84$. A different approach is presented in [10] where it is claimed that a constant space is sufficient. However, in order to obtain the constant space space, input objects (from Δ) are allowed to create other objects (from Γ) leading to a different and more powerful formalism than the one used by us in this paper.

References

1. Aman, B., Ciobanu, G.: Describing the immune system using enhanced mobile membranes. Electron. Notes Theoret. Comput. Sci. **194**, 5–18 (2008)
2. Aman, B., Ciobanu, G.: Turing completeness using three mobile membranes. In: Calude, C.S., Costa, J.F., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) UC 2009. LNCS, vol. 5715, pp. 42–55. Springer, Heidelberg (2009)
3. Aman, B., Ciobanu, G.: Mobility in Process Calculi and Natural Computing. Natural Computing Series. Springer, New York (2011)
4. Besozzi, D., Ciobanu, G.: A P system description of the Sodium-Potassium pump. In: Mauri, G., Păun, G., Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 210–223. Springer, Heidelberg (2005)
5. Bonchiş, C., Ciobanu, G., Izbaşa, C.: Encodings and arithmetic operations in membrane computing. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 621–630. Springer, Heidelberg (2006)
6. Cavaliere, M.: Evolution-communication P systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 134–145. Springer, Heidelberg (2003)

7. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, New York (2006)
8. Krishna, S.N., Rama, R.: P systems with replicated rewriting. J. Automata Lang. Comb. **6**(3), 345–350 (2001)
9. Leporati, A., Gutiérrez-Naranjo, M.A.: Solving subset sum by spiking neural P Systems with pre-computed resources. Fundamenta Informaticae **87**(1), 61–77 (2008)
10. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Constant-space P systems with active membranes. Fundamenta Informaticae **134**(1–2), 111–128 (2014)
11. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes. J. Automata Lang. Comb. **19**(1–4), 173–184 (2014)
12. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Nat. Comput. **10**, 613–632 (2011)
13. Păun, G.: P systems with active membranes: attacking NP-complete problems. J. Automata Lang. Comb. **6**, 75–90 (2001)
14. Păun, G.: Further Twenty Six Open Problems in Membrane Computing. In: Gutiérrez, M.A., et al. (eds.) Third Brainstorming Week on Membrane Computing, pp. 249–262, Fénix Editora, Sevilla (2005)
15. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
16. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity-membrane division, membrane creation. In: [15], pp. 302–336
17. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) CMC 2012. LNCS, vol. 7762, pp. 342–357. Springer, Heidelberg (2013)