Cell-like P systems with polarizations and minimal rules

Linqiang Pan [a], David Orellana-Martín [b], Bosheng Song [c,*], Mario J. Pérez-Jiménez [b]

[a] *Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China*

[b] *Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain*

[c] *College of Information Science and Engineering, Hunan University, Changsha 410082, Hunan, China*

Keywords:
Bio-inspired computing
Membrane computing
Minimal rule
Universality
PSPACE

A B S T R A C T

P systems with active membranes are a class of computation models in the area of membrane computing, which are inspired from the mechanism by which chemicals interact and cross cell membranes. In this work, we consider a normal form of P systems with active membranes, called cell-like P systems with polarizations and minimal rules, where rules are minimal in the sense that an object evolves to exactly one object with the application of an evolution rule or a communication rule, or an object evolves to two objects that are assigned to the two new generated membranes by applying a division rule. The present work investigates the computational power of P systems with polarizations and minimal rules. Specifically, results about Turing universality and non-universality are obtained with the combination of the number of membranes, the number of polarizations, and the types of rules. We also show that polarizationless P systems with minimal rules are equivalent to Turing machines working in a polynomial space, that is, the class of problems that can be solved in polynomial time by polarizationless P systems with minimal rules is equal to the class **PSPACE**.

1. Introduction

A cell is the basic unit of biological organization that constitutes all living organisms. Motivated by the behavior of a single cell and multiple cells cooperating in a certain environment, a lively branch of natural computing, called *membrane computing*, was proposed in 1998, and the literature of membrane computing has been growing fast covering theoretical results [44,48,53,62], applications on real-life problems [40,41,64], and implementation of computation models [21,24]. The computation models in membrane computing are called *P systems*, which have two main families (identified by the membrane structure): *cell-like P systems* (a rooted tree) [36] and *tissue-like P systems* [23,32] (a graph) or *neural-like P systems* [14,46,47] (a directed graph). For a more detailed presentation of membrane computing the reader can refer to [39], while a complete bibliography of this area is available on the web site http://ppage.psystems.eu.

Cell-like P systems have a hierarchical arrangement of membranes embedded in a main membrane, called the *skin* membrane [36]. The range of each membrane is defined as a compartment (region), which contains multisets of objects,

* Corresponding author.
E-mail addresses: lqpan@mail.hust.edu.cn (L. Pan), dorellana@us.es (D. Orellana-Martín), boshengsong@hnu.edu.cn (B. Song), marper@us.es (M.J. Pérez-Jiménez).

evolution rules and possibly other membranes. The outside region of the skin membrane is called *environment*. A membrane with no lower neighbors is called *elementary*; otherwise, the membrane is called *non-elementary*. In cell-like P systems, multisets of objects in different regions may evolve to other objects [36] or they can move between two regions [12,35]. With different biological, mathematical, or computer science motivations, the applications of rules in cell-like P systems are investigated in various ways, such as minimal parallelism [5,10], time-freeness [4,49,52], flat maximal parallelism [9,30,51, 61].

P systems with active membranes were proposed in [37], which have the important feature that each membrane has an electrical charge, that may be + (positive), − (negative), or 0 (neutral). The polarization of a membrane can be modified at a computation step. P systems with active membranes have five types of rules: (a) object evolution rules (an object evolves to a multiset of objects in a region); (b) *in* communication rules (an object enters a membrane); (c) *out* communication rules (an object exits a membrane); (d) dissolving rules (initiated by an object, a membrane is dissolved: all the objects in this membrane are released to the immediately outside region, and the rules in this dissolved membrane are lost); (e) division rules (initiated by an object, a membrane is divided into two membranes).

As computing devices, an important issue of P systems with active membranes is to study the computational power [8,11]. The first result about the universality of P systems with active membranes was presented in [34], and the result about the universality of P systems with active membranes was improved by optimizing the number of membranes, the types of rules and the number of polarizations [38]. The computational power of P systems with active membranes was also investigated with the consideration of various strategies for rule application, such as asynchronous [11] or minimal parallelism [5].

Another interesting issue of P systems with active membranes is to investigate computational efficiency, since such systems offer the attractive possibility of solving **NP**-complete problems in feasible time [6,22,28,37], such as the `SAT` problem [2,37,59,60,63], `Subset Sum` [43], and `Vertex Cover` [15]. Moreover, **PSPACE**-complete problems can also be solved by P systems with active membranes [27,54]. It is also known that P systems with active membranes without using membrane dissolving rules can solve the `QSAT` problem in linear time [1]. In [3], it is shown that the `QSAT` problem was solved by P systems with active membranes without polarizations and using evolution rules, send-out communication rules, dissolution rules and division for elementary and non-elementary membranes.

The computational complexity of (tissue) P systems with membrane division or membrane separation has also been investigated. In [58], it was proved that P systems with active membranes characterize **PSPACE**, that is, the class of problems solvable by P systems with active membranes is equal to **PSPACE** (further studied in [16–19,57]). In [55,56], an upper bound of the computational power of tissue P systems with cell separation or cell division is provided, respectively; it was thus shown that the class of problems that are solved in polynomial time by tissue P systems with cell separation or cell division is contained in the class **PSPACE**.

The present work considers a normal form of P systems with active membranes, called *cell-like P systems with minimal rules*, where the rules are minimal in the sense that an object evolves to exactly one object with the application of an evolution rule or a communication rule, or an object evolves to two objects that are assigned to the two new generated membranes by applying a division rule. More specifically, we consider six types of rules in cell-like P systems with minimal rules: (1) possibly modifying an object in a region but not moving it out of the region; (2) possibly modifying an object out of a region but not moving it into the region; (3) sending out and possibly modifying an object; (4) sending in and possibly modifying an object; (5) interchanging and possibly modifying two objects; (6) dividing elementary or non-elementary membranes.

The contributions of this work are summarized as follows:

(a) A normal form of P systems with active membranes is proposed, where the rules are minimal. Moreover, the environment as an important component is considered in cell-like P systems with minimal rules, and it is supposed that the objects initially located in the environment have an arbitrarily large number of copies.
(b) The computational power of cell-like P systems with polarizations and minimal rules (CPPMRs, for short) is investigated. It is proved that CPPMRs with arbitrarily many membranes, using rules of types (1), (2) or using rules of types (3), (5) or using rules of type (4) are able to compute the lengths of the strings of finite languages. We further prove that CPPMRs with four membranes, using rules of types (1), (2), (4) or using rules of types (3), (4) are Turing universal; the result about the universality can also be obtained for cell-like P systems with two membranes, with two polarizations and using rules of types (4), (5).
(c) The computational complexity of cell-like P systems with minimal rules and without polarizations (CPMRs, for short) is also investigated. It is shown that the computational power of CPMRs is equivalent to that of Turing machines working in polynomial space, that is, the class of problems solved by CPMRs in polynomial time is equal to the class of problems **PSPACE**. Specifically, on the one hand, CPMRs can solve the **PSPACE**-complete problem `QSAT`; on the other hand, Turing machines can simulate CPMRs in polynomial space.

The rest of the paper is organized as follows. The definitions of CPPMRs and recognizer CPPMRs are presented in section 2. The computational power of CPPMRs is investigated in section 3. In section 4, a characterization of **PSPACE** using CPMRs is given. Finally, conclusions and future works are presented in section 5.

2. Cell-like P systems with polarizations and minimal rules

In this section, we directly give the definitions of CPPMRs and recognizer CPPMRs. For the basic notions used in this work from formal language theory, one can refer to [45].

Definition 1. A cell-like P system with polarizations and minimal rules of degree $q \geq 1$ (CPPMR, for short) is a tuple

$$\Pi = (\Gamma, H, \mu, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out}),$$

where

- $\Gamma, \mathcal{E}$ are two alphabets of objects, where objects in $\mathcal{E}$ ($\mathcal{E} \subseteq \Gamma$) are initially present in the environment in an infinite number of copies;
- H is a finite set of labels;
- μ is an initial membrane structure;
- $\mathcal{M}_i$, $1 \leq i \leq q$, are initial multisets of objects placed in each membrane;
- $\mathcal{R}$ is a finite set of rules:
  (1) $[\ a\ ]_i^{\alpha_1} \to [\ b\ ]_i^{\alpha_2}$, $i \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in \Gamma$.
  (A simple object evolves in membrane i under the control of the charge of the membrane i. In this process the membrane polarization may be modified.)
  (2) $a[\ \ ]_i^{\alpha_1} \to b[\ \ ]_i^{\alpha_2}$, $i \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in \Gamma$.
  (A simple object evolves in the father region of membrane i under the control of the charge of the membrane i. In this process the membrane polarization may be modified. Note that one object and one membrane can be subject of only one rule.)
  (3) $[\ a\ ]_i^{\alpha_1} \to b[\ \ ]_i^{\alpha_2}$, $i \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in \Gamma$.
  (Send out of an object from the membrane i under the control of the charge of the membrane i. In this process, both the object and the membrane polarization may be modified.)
  (4) $a[\ \ ]_i^{\alpha_1} \to [\ b\ ]_i^{\alpha_2}$, $i \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in \Gamma$.
  (Send an object into the membrane i under the control of the charge of the membrane i. In this process, both the object and the membrane polarization may be modified. Note that one object and one membrane can be subject of only one rule.)
  (5) $a[\ b\ ]_i^{\alpha_1} \to c[\ d\ ]_i^{\alpha_2}$, $i \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b, c, d \in \Gamma$.
  (Under the control of the charge of the membrane i, an object placed in the father region of membrane i is sent into the membrane, possibly modified, and simultaneously, an object placed in membrane i is sent out. In this process, all objects and the membrane polarization may be modified.)
  (6) $[\ a\ ]_i^{\alpha_1} \to [\ b\ ]_i^{\alpha_2}[\ c\ ]_i^{\alpha_3}$, $i \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in \Gamma$.
  (A weak division rule is applicable if membrane i has the polarization α_1 and contains object a; moreover, membrane i cannot be the skin membrane or the output membrane. Under the control of object a and the polarization of membrane i, such membrane is divided into two membranes: the object specified in the rule is replaced in the two new membranes, the polarizations of the two new membranes can be modified, all the other objects are duplicated in the two generated membranes. We remark that if the division membrane is a non-elementary membrane, then all membranes and objects included in this non-elementary membrane will be duplicated in each of the new generated membranes.)
- $i_{out} \in \{0, 1, \ldots, q\}$ is the output region.

A CPPMR works in the maximally parallel manner [36]. Note that any object can be subject to only one rule of any type, and any membrane can be subject to only one rule of any type.

A *configuration* of a CPPMR is described by the current membrane structure, the current objects in each region, and the current multiset of objects over $\Gamma \setminus \mathcal{E}$ in the environment. By using rules described above, one can define *transitions* among configurations. Any sequence of transitions starting from the initial configuration is called a *computation*. A halting configuration is such that no rule can be used in the system. The result of a halting computation is the number of specified objects present in the output region when a computation halts. A non-halting computation produces no result.

We denote by $N(\Pi)$ the set of natural numbers generated by system Π. The family of all sets $N(\Pi)$ of natural numbers computed by systems Π with at most m membranes, with at most k polarizations, and using rules as in *list-of-rules* is denoted by $NOP_m(char_k, list\text{-}of\text{-}rules)$. If parameter m is not bounded then it is replaced by $*$.

In this work, we use recognizer CPPMRs to solve decision problems, one can consult Chapter 12 in [39,42] for details.

Definition 2. A recognizer cell-like P system without polarizations and minimal rules of degree $q \geq 1$ (recognizer CPMR, for short) is a tuple

$$\Pi = (\Gamma, H, \Sigma, \mu, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out}),$$

such that:

- $(\Gamma, H, \mu, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ is a cell-like P system with polarizations and minimal rules of degree $q \geq 1$;
- Γ has two distinguished objects `yes` and `no`, with at least one copy of them present in some multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$, but none of them present in $\mathcal{E}$;
- Σ is an (input) alphabet strictly contained in Γ, and such that $\mathcal{E} \subseteq \Gamma \setminus \Sigma$;
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over $\Gamma \setminus \Sigma$;
- $i_{in} \in \{1, \ldots, q\}$ is the input membrane, and $i_{out} = 0$;
- all computations halt;
- if $\mathcal{C}$ is a computation of Π, then either object `yes` or object `no` (but not both) must have been released into the environment, and only at the last step of the computation.

If all computations of CPMR with the same input always produce the same result, then such CPMR is called *confluent*.

Definition 3. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer P systems from **CPMR** in a uniform way, if the following conditions hold:

- the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$;
- there exists a pair (cod, s) of polynomial-time computable functions over I_X such that:
  - for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;
  - for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;
  - the family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and it performs at most $p(|u|)$ steps;
  - the family $\mathbf{\Pi}$ is sound with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;
  - the family $\mathbf{\Pi}$ is complete with regard to (X, cod, s), that is, for each $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting one.

The set of all decision problems that can be solved by recognizer CPMRs in a uniform way and polynomial time is denoted by $\mathbf{PMC_{CPMR}}$.

3. Computational power of CPPMRs

In this section, we investigate the computational power of CPPMRs. We denote by $NFIN$ the family of all finite sets of positive integers, and by NRE the family of sets of numbers which are Turing computable.

Theorem 3.1. $NOP_*(char_3, (1), (2)) \subseteq NFIN$.

Proof. Obviously, by using rules of types (1) and (2), an object can evolve to another object in a membrane or outside of a membrane, and the system cannot bring objects into membranes from the environment. Therefore, in any computation, the number of objects in the CPPMRs will not change by using rules of types (1) and (2). So, the number of configurations reachable by computations is finite. Therefore, the system can only generate finite sets of natural numbers. □

Theorem 3.2. $NOP_*(char_3, (4)) \subseteq NFIN$.

Proof. According to rules of type (4), an object can be sent into a membrane, and the membrane polarization may be modified. We note that only rules of type (4) exist in a CPPMR, hence the number of objects can be increased if and only if the objects placed in the environment are sent into the skin membrane. Without loss of generality, it is assumed that the skin membrane has neutral polarization. When a rule of type (4) is used, we have the following two cases: (i) the object entering into the skin membrane does not belong to $\mathcal{E}$; (ii) the object entering into the skin membrane belongs to $\mathcal{E}$.

- Case (i): Obviously, the number of objects from the environment (that is, from $\Gamma \setminus \mathcal{E}$) is finite, so the number of objects entering into the system by using the rules of type (4) is finite.
- Case (ii): When a sequence of (always different) objects is sent into the system, the polarization of the skin membrane must be changed at the step that the last object of the sequence is sent into the system, otherwise, these rules can be applied forever, and the computation never stops. Therefore, a finite number of objects can enter the systems.

In general, the number of objects in the system in each configuration is finite. So, the number of reachable configurations is finite and the generated set of numbers by the system is finite. Therefore, the theorem holds. □

Theorem 3.3. $NOP_*(char_3, (3), (5)) \subseteq NFIN$.

Proof. By using rules of type (5), an object in a membrane is exchanged with an object out of this membrane. By applying rules of type (3), an object is sent out of a membrane. The number of objects in the system cannot be increased in any computation starting from the initial configuration to a halting configuration. Hence the number of configurations of the system is finite, and the set of natural numbers generated by the system is finite. Therefore, the theorem holds. □

In what follows, we will show some results about the Turing universality of CPPMRs, which are proved by simulating register machines.

A register machine is a tuple $M = (m, H, l_0, l_h, I)$, where m is a natural number (it represents the number of registers), H is a set of labels, $l_0, l_h \in H$ are initial and halting one, respectively, I is a set of instructions $l_i : (op(r), l_j, l_k)$ ($op(r)$ is an ADD operation or a SUB operation on register r of M) of the following forms:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (when this instruction is applied, the value of register r is increased by one and then the machine will carry out one of the instructions with labels l_j, l_k);
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (when this instruction is applied, we have two cases: if the content of register r is greater than zero, then the content of register r is decreased by one and the machine will carry out the instruction with label l_j; otherwise, the content of register r is not changed and the machine will carry out the instruction with label l_k);
- $l_h : \mathtt{HALT}$ (the halt instruction).

A register machine M generates the set of numbers $N(M)$ as follows: initially all registers are empty, M starts with the initial instruction l_0, by applying instructions as indicated by the labels, the register machine will halt if and when it reaches the halting instruction l_h. The number stored in the first register at the halting moment is called the number generated by the register machine. It is known that register machines and Turing machines are equivalent, that is, both of them characterize NRE [25].

Theorem 3.4. $NOP_4(char_3, (3), (4)) = NRE$.

Proof. It is known that a register machine with three registers is universal [7,25], so we assume that the simulated register machine $M = (m, H, l_0, l_h, I)$ has $m = 3$. We construct a CPPMR Π that simulates M.

$$\Pi = (\Gamma, H, \mu, \mathcal{E}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4, \mathcal{R}, 1),$$

where

$$\Gamma = \{l, l', l'' \mid l \in H\} \cup \{a, a'\},$$
$$H = \{1, 2, 3, 4\},$$
$$\mu = [\,[\,]_1^0[\,]_2^0[\,]_3^0\,]_4^0,$$
$$\mathcal{E} = \{a\},$$
$$\mathcal{M}_1 = \mathcal{M}_2 = \mathcal{M}_3 = \emptyset,\ \mathcal{M}_4 = \{l_0\},$$

and the set $\mathcal{R}$ of rules is defined as follows (each register $r = 1, 2, 3$ of M corresponds to a membrane labeled with r, the value of register r is represented as the number of copies of object a in membrane r. When object l_i appears in membrane 4, the system starts to simulate the instruction l_i. The number of copies of object a in membrane 1 is considered the result of a computation).

- We construct the following rules to simulate an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ of M:

$$r_1 : l_i[\,]_r^0 \to [\,l_i\,]_r^0,$$
$$r_2 : [\,l_i\,]_r^0 \to l_i'[\,]_r^+,$$
$$r_3 : [\,l_i'\,]_4^0 \to l_i'[\,]_4^+,$$
$$r_4 : a[\,]_4^+ \to [\,a\,]_4^0,$$
$$r_5 : a[\,]_r^+ \to [\,a\,]_r^0,$$
$$r_6 : l_i'[\,]_4^0 \to [\,l_j\,]_4^0,$$
$$r_7 : l_i'[\,]_4^0 \to [\,l_k\,]_4^0.$$

At the first two steps, object l_i evolves to l_i' and the polarization of membrane r is changed from neutral to positive. With the polarization of membrane 4 changes to positive, one copy of a is sent into membrane 4, returning the polarization of membrane 4 to neutral. At step 5, object a is sent into the membrane r changing the polarization back to neutral (the number of copies of a in membrane r is increased by one, which simulates that the number stored in register r is increased by one); in parallel, object l_j or l_k appears in membrane 4 non-deterministically by using rule r_6 or r_7. Therefore, the next instruction with label l_j or l_k will be simulated at the next step.

- We construct the following rules to simulate a SUB instruction $l_i : (\mathtt{SUB}(r),\ l_j, l_k)$ of M:

$$r_8 : l_i[\ ]_r^0 \to [\ l_i\ ]_r^0,$$
$$r_9 : [\ l_i\ ]_r^0 \to l_i'[\ ]_r^-,$$
$$r_{10} : [\ a\ ]_r^- \to a'[\ ]_r^+,$$
$$r_{11} : [\ l_i'\ ]_4^0 \to l_i'[\ ]_4^0,$$
$$r_{12} : l_i'[\ ]_4^0 \to [\ l_i''\ ]_4^0,$$
$$r_{13} : l_i''[\ ]_r^+ \to [\ l_i''\ ]_r^+,$$
$$r_{14} : l_i''[\ ]_r^- \to [\ l_i''\ ]_r^-,$$
$$r_{15} : [\ l_i''\ ]_r^+ \to l_j[\ ]_r^0,$$
$$r_{16} : [\ l_i''\ ]_r^- \to l_k[\ ]_r^0.$$

The simulation of a SUB instruction l_i works as follows. At the first two steps, object l_i evolves to l_i' and the polarization of membrane r is changed to negative. In what follows, the system will check whether membrane r contains object a.

- Membrane r contains at least one copy of object a (that is, the number stored in register r is greater than 0). One copy of object a evolves to a', which is sent out of membrane r, and the polarization of such membrane is changed to positive; in membrane 4, object l_i' evolves to l_i'' in two steps. By using rules r_{13} and r_{15}, object l_i'' evolves to l_j, returning the polarization of membrane r to neutral. Hence one copy of object a is consumed in membrane r (simulating that the number stored in register r is decreased by one), and the next instruction l_j will be simulated at the next step.
- Membrane r has no object a (that is, the number stored in register r is 0). Object l_i' will be evolved to l_i'' by using rules r_{11}, r_{12}. At the next two steps, object l_i'' evolves to l_k, and the polarization of membrane r is returned neutral. Hence, the next instruction l_k will be simulated at the next step.

Hence, system Π simulates the SUB instruction of M correctly.

The computation can stop only when object l_h appears in membrane 4 at some step. The number of copies of object a in membrane 1 is the result of the system, hence $N(M) = N(\Pi)$, which concludes the proof. □

Theorem 3.5. $NOP_4(char_3, (1), (2), (4)) = NRE$.

Proof. For a register machine $M = (3, H, l_0, l_h, I)$ with three registers, we construct the CPPMR, Π, to simulate M.

$$\Pi = (\Gamma, H, \mu, \mathcal{E}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4, \mathcal{R}, 1),$$

where

- $\Gamma = \{l, l', l'' \mid l \in H\} \cup \{a, a'\}$,
- $H = \{1, 2, 3, 4\}$,
- $\mu = [\ [\ \ ]_1^0[\ \ ]_2^0[\ \ ]_3^0\ ]_4^0$,
- $\mathcal{E} = \{a\}$,
- $\mathcal{M}_1 = \mathcal{M}_2 = \mathcal{M}_3 = \emptyset$, $\mathcal{M}_4 = \{l_0\}$,

and the rule set $\mathcal{R}$ is defined as follows (each register $r = 1, 2, 3$ of M corresponds to a membrane labeled with r, the value of register r is represented as the number of copies of a in membrane r. The system starts to simulate the instruction l_i when object l_i appears in membrane 4. The result of the computation is encoded as the number of copies of object a in membrane 1).

- The following rules are used to simulate an ADD instruction $l_i : (\mathtt{ADD}(r), l_j,\ l_k)$ of M:

$$r_1 : l_i[\ ]_r^0 \to l_i'[\ ]_r^+,$$

$$r_2 : [\, l_i' \,]_4^0 \to [\, l_i'' \,]_4^+,$$
$$r_3 : a[\ \ ]_4^+ \to [\, a \,]_4^0,$$
$$r_4 : a[\ \ ]_r^+ \to [\, a \,]_r^0,$$
$$r_5 : [\, l_i'' \,]_4^0 \to [\, l_j \,]_4^0,$$
$$r_6 : [\, l_i'' \,]_4^0 \to [\, l_k \,]_4^0.$$

At the first two steps, object l_i evolves to l_i'' in membrane 4, and the polarizations of membrane r and membrane 4 are changed to positive. At the next two steps, one copy of object a is sent into membrane r from the environment (simulating that the value of register r is increased by one) and the polarizations of membrane r and membrane 4 are returned to neutral by using rules r_3, r_4. In parallel, at step 4, object l_j or l_k appears in membrane 4 non-deterministically by using rule r_5 or r_6. Therefore, the next instruction with label l_j or l_k will be simulated at the next step.

- We construct the following rules to simulate a SUB instruction $l_i : (\mathtt{SUB}(r),\ l_j, l_k)$ of M:

$$r_7 : l_i[\ \ ]_r^0 \to l_i'[\ \ ]_r^-,$$
$$r_8 : [\, a \,]_r^- \to [\, a' \,]_r^+,$$
$$r_9 : [\, l_i' \,]_4^0 \to [\, l_i'' \,]_4^0,$$
$$r_{10} : l_i''[\ \ ]_r^+ \to l_j[\ \ ]_r^0,$$
$$r_{11} : l_i''[\ \ ]_r^- \to l_k[\ \ ]_r^0.$$

At the first step, object l_i evolves to l_i', changing the polarization of membrane r to negative by using rule r_7. In what follows, the system will check whether membrane r contains object a.

- Membrane r contains at least one copy of object a (that is, the number stored in register r is greater than 0). With membrane r having negative polarization, one copy of object a evolves to a' in membrane r, and the polarization of membrane r is changed to positive; in addition, object l_i' in membrane 4 evolves to l_j at the next two steps by using rules r_9, r_{10}, and the polarization of membrane r is returned to neutral. Hence one copy of object a is consumed in membrane r (simulating that the value of register r is decreased by one), and the next instruction l_j will be simulated at the next step.
- Membrane r has no object a (that is, the number stored in register r is 0). At the next two steps, rules r_9, r_{11} are used one by one, object l_i' evolves to l_k in membrane 4, returning the polarization of membrane r to neutral. Hence, the next instruction l_k will be simulated at the next step.

Hence, system Π correctly simulates the SUB instruction of M.

The computation can stop only when object l_h appears in membrane 4 at some step. The number of copies of object a in membrane 1 is the result of the system, hence $N(M) = N(\Pi)$, which concludes the proof. □

Theorem 3.6. $NOP_2(char_2, (4), (5)) = NRE$.

Proof. For a register machine $M = (3, H, l_0, l_h, I)$ with three registers, we construct a CPPMR, Π, to simulate M.

$$\Pi = (\Gamma, H, \mu, \mathcal{E}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, 2),$$

where

- $\Gamma = \{a_i, a_i' \mid 1 \le i \le 3\} \cup \{l, l', l'', l''', l^{iv} \mid l \in H\}$,
- $H = \{1, 2\}$,
- $\mu = [\ [\ \ ]_2^0\ ]_1^0$,
- $\mathcal{E} = \{a_i \mid 1 \le i \le 3\} \cup \{l^{iv} \mid l \in H\}$,
- $\mathcal{M}_1 = \{l_0\}$, $\mathcal{M}_2 = \emptyset$,

and the set $\mathcal{R}$ of rules is defined as follows (the result of the computation is encoded as the number of copies of object a_1 in membrane 2. When object l_i appears in membrane 1, the system starts to simulate the instruction l_i).

- We construct the following rules to simulate an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ of M:

$$r_1 : a_r[\, l_i \,]_1^0 \to l_i'[\, a_r \,]_1^0,$$

$$r_2 : a_r[\ ]_2^0 \to [\ a_r\ ]_2^0,$$
$$r_3 : l_i'[\ ]_1^0 \to [\ l_j\ ]_1^0,$$
$$r_4 : l_i'[\ ]_1^0 \to [\ l_k\ ]_1^0.$$

An ADD instruction l_i is simulated in two steps. First, object l_i in membrane 1 is exchanged with object a_r from the environment, then object l_i evolves to l_i'. At step 2, object a_r is sent into membrane 2 (simulating that the number stored in register r is increased by one); in parallel, object l_j or l_k appears in membrane 1 non-deterministically by using rule r_3 or r_4. Therefore, the next instruction with label l_j or l_k will be simulated at the next step.

- We construct the following rules to simulate each SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ of M:

$$r_5 : l_i^{iv}[\ l_i\ ]_1^0 \to l_i'[\ l_i^{iv}\ ]_1^0,$$
$$r_6 : l_i^{iv}[\ a_r\ ]_2^0 \to a_r'[\ l_i^{iv}\ ]_2^-,$$
$$r_7 : l_i'[\ ]_1^0 \to [\ l_i'\ ]_1^0,$$
$$r_8 : l_i'[\ ]_2^- \to [\ l_i''\ ]_2^0,$$
$$r_9 : l_i'[\ ]_2^0 \to [\ l_i'''\ ]_2^0,$$
$$r_{10} : a_r'[\ l_i''\ ]_2^0 \to l_j[\ a_r'\ ]_2^0,$$
$$r_{11} : l_i^{iv}[\ l_i'''\ ]_2^0 \to l_k[\ l_i^{iv}\ ]_2^0.$$

The simulation of a SUB instruction l_i works as follows. At the first step, object l_i in membrane 1 is exchanged with object l_i^{iv}, then object l_i evolves to l_i'. In what follows, the system will check whether membrane 2 contains object a_r.

- Membrane 2 contains at least one copy of object a_r (that is, the value stored in register r is greater than 0). With object l_i^{iv} present in membrane 1, rule r_6 is applied, one copy of object a_r evolves to a_r', and object l_i^{iv} is sent into membrane 2; in parallel, object l_i' evolves to l_i'', which is sent into membrane 2 by applying rules r_7, r_8, returning the polarization of membrane 2 to neutral. At step 4, object l_i'' evolves to l_j, which is sent out of membrane 2. Hence one copy of object a_r is consumed in membrane 2 (it simulates that the number stored in register r is decreased by one), and the next instruction l_j will be simulated at the next step.
- Membrane 2 has no object a_r (that is, the value stored in register r is 0). At the next two steps, object l_i' evolves to l_i''', which is sent into membrane 2. At step 4, by applying rule r_{11}, object l_i''' evolves to l_k, which is sent out of membrane 2. Hence, the next instruction l_k will be simulated at the next step.

Hence, system Π correctly simulates the SUB instruction of M.

The computation can stop only when object l_h appears in membrane 4 at some step. The number of copies of object a_1 in membrane 2 is the result of the computation, hence $N(M) = N(\Pi)$. □

4. CPMRs characterize PSPACE

In this section, we first construct a family of recognizer CPMRs to solve the QSAT problem, then we will show that recognizer CPMRs can be simulated by a Turing machine working in a polynomial space.

4.1. Solving the QSAT problem by CPMRs

Given a Boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, with Boolean variables $x_1, \ldots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ (where Q_n is $\exists$ if n is odd, and Q_n is $\forall$ otherwise) is said to be the (existential) *fully quantified* formula associated with $\varphi(x_1, \ldots, x_n)$. We say that φ^* is satisfiable if there exists a truth assignment, σ, over $\{i : 1 \le i \le n \wedge i \text{ odd}\}$ such that each extension, σ^* (an extension of SAT models for QBF), of σ over $\{1, \ldots, n\}$ verifies $\sigma^*(\varphi(x_1, \ldots, x_n)) = 1$. For further detailed description of the QSAT problem, one can refer to [1,33,50,57].

We encode the instance φ as follows, which will be applied as input to the CPMR:

$$cod(\varphi) = \alpha_{1,1} \ldots \alpha_{1,m}\ \alpha_{2,1} \ldots \alpha_{2,m} \ldots\ldots \alpha_{n,1} \ldots \alpha_{n,m},$$

where m is the number of clauses, and for $1 \le i \le n, 1 \le j \le m$.

$$\alpha_{i,j} = \begin{cases} d_{i,j} & \text{if clause } C_j \text{ contains variable } x_i; \\ d'_{i,j} & \text{if clause } C_j \text{ contains variable } \neg x_i; \\ d''_{i,j} & \text{if clause } C_j \text{ does not contain variable } x_i \text{ and variable } \neg x_i. \end{cases}$$

Consider the primitive recursive and bijective function from $\mathbb{N}^2$ onto $\mathbb{N}$ defined by $\langle n, m\rangle = ((n+m)(n+m+1)/2)+n$. For each $t = \langle n, m\rangle$, we define the recognizer CPMR $(\Pi(t) = (\Gamma, H, \mu, \mathcal{E}, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_{n+2}, \mathcal{R}, i_{in}, i_{out})$ as follows:

- $\Gamma = \Sigma \cup \{a_i, t_{i,m+1}, f_{i,m+1} \mid 1 \le i \le n\} \cup \{t_{i,j}, f_{i,j} \mid 1 \le i \le n, 1 \le j \le m\} \cup \{c_j, e_j, e'_j \mid 1 \le j \le m\} \cup \{\alpha_i \mid 0 \le i \le n^2 + 2nm + n + m + 4\} \cup \{d, e_{m+1}, e'_{m+1}, g, g', p, s, t, t', \mathtt{yes}, \mathtt{no}\}$,
- $H = \{1, 2, \ldots, n+2\}$,
- $\mu = [\,[\,[\,\ldots[\,[\,]_{n+1}\,]_n \cdots\,]_2\,]_1\,]_{n+2}$,
- $\Sigma = \{d_{i,j}, d'_{i,j}, d''_{i,j} \mid 1 \le i \le n, 1 \le j \le m\}$,
- $\mathcal{E} = \emptyset$,
- $\mathcal{M}_1 = \{a_1, g, p\}$, $\mathcal{M}_i = \{g\}$, $2 \le i \le n-1$, $\mathcal{M}_n = \mathcal{M}_{n+1} = \emptyset$, $\mathcal{M}_{n+2} = \{\alpha_1, g\}$,
- $i_{in} = n+1$ is the input membrane,
- $i_{out} = 0$ is the environment,
- the set $\mathcal{R}$ consists of the following rules:

$$
\begin{aligned}
&r_{1,i} : [\,a_i\,]_i \to [\,t_{i,1}\,]_i[\,f_{i,1}\,]_i,\ 1 \le i \le n,\\
&r_{2,i,j} : t_{i,1}[\,\,]_j \to [\,t_{i,1}\,]_j, 1 \le i \le n-1, i < j \le n,\\
&r_{3,i,j} : f_{i,1}[\,\,]_j \to [\,f_{i,1}\,]_j,\ 1 \le i \le n-1, i < j \le n,\\
&r_{4,i,j} : t_{i,j}[\,d_{i,j}\,]_{n+1} \to c_j[\,t_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{5,i,j} : t_{i,j}[\,d'_{i,j}\,]_{n+1} \to d[\,t_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{6,i,j} : t_{i,j}[\,d''_{i,j}\,]_{n+1} \to d[\,t_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{7,i,j} : f_{i,j}[\,d_{i,j}\,]_{n+1} \to d[\,f_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{8,i,j} : f_{i,j}[\,d'_{i,j}\,]_{n+1} \to c_j[\,f_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{9,i,j} : f_{i,j}[\,d''_{i,j}\,]_{n+1} \to d[\,f_{i,j+1}\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m,\\
&r_{10,i,j} : [\,t_{i,j+1}\,]_{n+1} \to t_{i,j+1}[\,\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m-1,\\
&r_{11,i,j} : [\,f_{i,j+1}\,]_{n+1} \to f_{i,j+1}[\,\,]_{n+1},\ 1 \le i \le n, 1 \le j \le m-1,\\
&r_{12,i} : [\,t_{i,m+1}\,]_{n+1} \to a_{i+1}[\,\,]_{n+1},\ 1 \le i \le n-1,\\
&r_{13,i} : [\,f_{i,m+1}\,]_{n+1} \to a_{i+1}[\,\,]_{n+1},\ 1 \le i \le n-1,\\
&r_{14,i,j} : [\,a_i\,]_j \to a_i[\,\,]_j,\ 2 \le i \le n-1, i < j \le n,\\
&r_{15} : [\,t_{n,m+1}\,]_{n+1} \to [\,e_1\,]_{n+1},\\
&r_{16} : [\,f_{n,m+1}\,]_{n+1} \to [\,e_1\,]_{n+1},\\
&r_{17,j} : c_j[\,e_j\,]_{n+1} \to e'_{j+1}[\,e_{j+1}\,]_{n+1},\ 1 \le j \le m,\\
&r_{18} : [\,e_{m+1}\,]_{n+1} \to t[\,\,]_{n+1},\\
&\{r_{19,i} : g[\,t\,]_i \to s[\,g'\,]_i,\\
&r_{20,i} : s[\,t\,]_i \to t[\,s'\,]_i \mid Q_i = \forall, 1 \le i \le n\},\\
&\{r_{21,i} : g[\,t\,]_i \to t[\,g'\,]_i \mid Q_i = \exists, 1 \le i \le n\},\\
&r_{22} : t[\,p\,]_1 \to \mathtt{yes}[\,t'\,]_1,\\
&r_{23} : [\,\mathtt{yes}\,]_{n+2} \to \mathtt{yes}[\,\,]_{n+2},\\
&r_{24,i} : [\,\alpha_i\,]_{n+2} \to [\,\alpha_{i+1}\,]_{n+2},\ 1 \le i \le n^2 + 2nm + n + m + 3,\\
&r_{25} : \alpha_{n^2+2nm+n+m+4}[\,p\,]_1 \to \mathtt{no}[\,\alpha_0\,]_1,\\
&r_{26} : [\,\mathtt{no}\,]_{n+2} \to \mathtt{no}[\,\,]_{n+2}.
\end{aligned}
$$

The computation of the P system $\Pi(\varphi^*)$ that solves the QSAT problem can be divided into four phases: generation phase, checking phase, quantifier phase, and output phase. In what follows, we explain how the system works in each phase.

Generation phase. A binary tree structure is produced by dividing non-elementary membranes with labels $1, 2, \ldots, n$; in this way the system produces all truth assignments for the variables $x_1, \ldots, x_n$. In addition, the system checks the clauses that are satisfied by the corresponding truth assignment. Therefore, some of the objects $c_1, c_2, \ldots, c_m$ appear in a

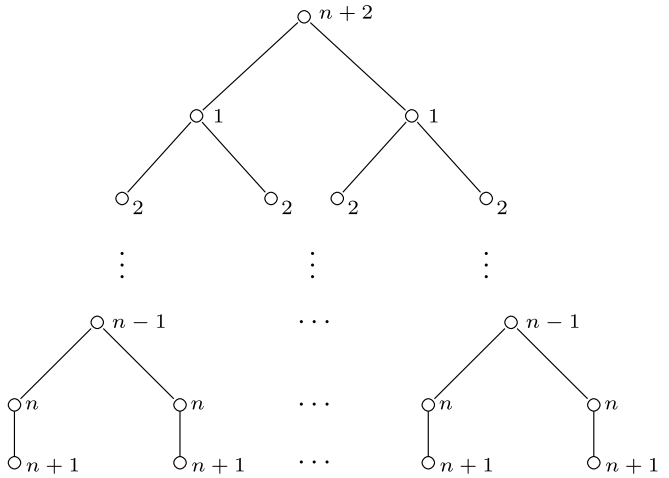

Fig. 1. The membrane structure of the system when the generation phase completes, where the numbers at nodes indicate the labels of membranes.

membrane n when the generation phase completes (object c_j corresponds to the clauses satisfied by the truth-assignment). In parallel in membrane $n+2$, the subscript of object α_i is increased in order to count the number of steps.

Next we describe how the system assigns and looks for the clauses satisfied by the truth-assignment of variable x_i.

Division rule $r_{1,i}$ is used when object a_i appears in membrane i, two copies of membrane with label i are generated, where objects $t_{i,1}$ and $f_{i,1}$ are produced in these two new membranes i. In the following steps, object $t_{i,1}$ (resp., $f_{i,1}$) will be sent into membrane n by using rules $r_{2,i,j}$ (resp., $r_{3,i,j}$).

The appearance of object $t_{i,1}$ (resp., $f_{i,1}$) in membrane n shows that the clauses made true by the truth-assignment of variable x_i will be checked. Specifically, if object $t_{i,j}$ appears in a membrane n and object $d_{i,j}$ (resp., $d'_{i,j}, d''_{i,j}$) appears in the immediate lower membrane n, object $d_{i,j}$ (resp., $d'_{i,j}$ and $d''_{i,j}$) evolves to c_j (resp., d), which is sent out of membrane $n+1$; in addition, object $t_{i,j}$ evolves to $t_{i,j+1}$, which is sent into membrane $n+1$. Similarly, if object $f_{i,j}$ appears in a membrane n and object $d'_{i,j}$ (resp., $d_{i,j}, d''_{i,j}$) appears in the immediate lower membrane n, object $d'_{i,j}$ (resp., $d_{i,j}$ and $d''_{i,j}$) evolves to c_j (resp., d), which is sent out of membrane $n+1$; in addition, object $f_{i,j}$ evolves to $f_{i,j+1}$, which is sent into membrane $n+1$. At the next step, object $t_{i,j+1}$ (resp., $f_{i,j+1}$) is sent out of membrane $n+1$. This process can be repeated until object $t_{i,m+1}$ (resp., $f_{i,m+1}$) appears in membrane $n+1$.

By using rule $r_{12,i}$ (resp., $r_{13,i}$), object $t_{i,m+1}$ (resp., $f_{i,m+1}$) evolves to a_{i+1}, which is sent out of membrane $n+1$. By using rule $r_{14,i,j}$, object a_{i+1} is sent out from membrane n to membrane $i+1$. When object a_{i+1} appears in membrane $i+1$, the system starts to assign and look for the clauses satisfied by the truth-assignment of variable x_{i+1}.

Rule r_{15} (resp., r_{16}) is applied only when the object $t_{n,m+1}$ (resp., $f_{n,m+1}$) appears in membrane $n+1$. Object $t_{n,m+1}$ (resp., $f_{n,m+1}$) evolves to e_1, and the generation phase completes.

We count the number of steps taken in this phase. For variable x_1, division rule takes one step, sending object $t_{1,1}$ or $f_{1,1}$ into membrane n takes $n-1$ steps, checking the clauses satisfied by variable x_1 takes $2m$ steps, sending object a_2 into membrane 2 costs $n-2$ steps. Hence the system takes $2n+2m-2$ steps to assign and look for the clauses satisfied by variable x_1. For variable x_2, this process takes $2n+2m-4$ steps. Note that objects $t_{2,1}$ and $f_{2,1}$ transfer to membrane n from membrane 2 (objects $t_{1,1}$ and $f_{1,1}$ transfer to membrane n from membrane 1), and object a_3 transfers to membrane 3 from membrane n (object a_2 transfers to membrane 2 from membrane n). Therefore, we can deduce that two steps are decreased when the system assigns and looks for the clauses satisfied by variable from x_i to x_{i+1} $(1 \le i \le n-2)$. So the total number of steps is $n^2+2nm-n-2m$. Hence the system takes $2m+1$ steps for assigning and looking for the clauses satisfied by variable x_n. So the generation phase takes $n^2+2nm-n+1$ steps. When the generation phase completes, the membrane structure is shown in Fig. 1.

Checking phase. We have the membrane structure as shown in Fig. 1 when the generation phase completes. Clearly, each membrane n contains a membrane $n+1$, which contains some objects from the set $\{c_1, c_2, \ldots, c_m\}$ (object c_j corresponds to the j-th clause satisfied by the truth assignment of the variables).

Object c_1 starts to be checked in each membrane n if membrane $n+1$ contains object e_1. If a membrane n has object c_1, then object e_1 evolves to e'_2, which is sent out of membrane $n+1$; in addition, object c_1 evolves to e_2, which is sent into membrane $n+1$. Now object c_2 starts to be checked in each membrane n. Note that if object c_j does not appear in a membrane n, then $r_{17,j}$ cannot be applied and the computation in that membrane n will stop.

When a membrane $n+1$ has the object e_{m+1}, it means the immediate upper membrane n has all the objects $c_1, c_2, \ldots, c_m$, so rule r_{18} is used, object e_{m+1} evolves to t, which is sent out of membrane $n+1$.

In general, the checking phase takes $m+1$ steps.

Quantifier phase. In this phase, we will check whether the formula φ^* is satisfied. Object t moves upwards the binary tree structure, at each level, checking one quantifier $\forall$ by rules $r_{19,i}, r_{20,i}$ or $\exists$ by rules $r_{21,i}$.

For quantifier $Q_i = \forall$, we only need to prove the following fact: one copy of object t appears in the upper level membrane if and only if there exists one copy of object t in each lower level membrane i. Specifically, by applying rule $r_{19,i}$, object g evolves to g', which is sent into a membrane i; in addition, object t evolves to s, which is sent out of membrane i. By using rule $r_{20,i}$, object s is exchanged with object t in another membrane i, one copy of object t is sent to the upper level membrane. Note that there is at most one copy of object t in a membrane i.

For quantifier $Q_i = \exists$, we only need to prove the following fact: one copy of object t appears in the upper level membrane if and only if there exists at least one copy of object t in lower level membrane i. Specifically, by using rule $r_{21,i}$, object g evolves to g', which is sent into a membrane i; and object t is sent out of membrane i.

Hence the quantifier phase takes at most $2n$ steps.

Output phase. When object t appears in membrane $n+2$, it means the formula φ^* is satisfiable. In this case, object t evolves to t', which is sent into membrane 1; and object p evolves to `yes`, which is sent out of membrane 1. Besides, the subscript of object α_i is increased by one in membrane $n+2$. At the next step, object `yes` is sent to the environment. At this moment, no rule is enabled in the system and the computation halts, so the result of the system is *affirmative*.

At step $n^2+2nm+n+m+3$, if membrane $n+2$ does not contain object t, it means the formula φ^* is not satisfiable. In this case, at step $n^2+2nm+n+m+3$, object $\alpha_{n^2+2nm+n+m+3}$ evolves to $\alpha_{n^2+2nm+n+m+4}$ in membrane $n+2$. Next, object $\alpha_{n^2+2nm+n+m+4}$ evolves to α_0, which is sent into membrane 1; and object p evolves to `no`, which is sent out of membrane 1. At step $n^2+2nm+n+m+5$, object `no` is sent to the environment. After that, no rule is enabled in the system and the computation halts, hence the answer of the system is *negative*.

The necessary resources of the constructed P system are of polynomial order: (1) the size of the set Γ: $n^2+7nm+4n+4m+16$; (2) the initial number of membranes: $n+2$; (3) the initial number of objects in membranes: $n+3$; (4) the number of rules: $(5n^2+20nm+5n+4m+18)/2$; (5) the maximum length of a rule: 4. Hence, a deterministic Turing machine exists, which builds the P system $\Pi(\varphi^*)$ in a polynomial time with respect to n and m.

Obviously, P system $\Pi(\varphi^*)$ with input multiset $cod(\varphi)$ takes at most $n^2+2nm+n+m+4$ steps when the output of the system is `yes` in the halting configuration, while the system takes at most $n^2+2nm+n+m+5$ steps when the output of the system is `no` in the halting configuration. Therefore, a polynomial bound for the number of steps of the computation exists.

Hence, the `QSAT` problem is solved by a uniform family of recognizer CPMRs in polynomial time. So, we get the following result:

Theorem 4.1. `QSAT` $\in$ **PMC$_{\text{CPMR}}$**.

As **PMC$_{\text{CPMR}}$** is closed under polynomial time reductions, we get the following result:

Corollary 4.1. PSPACE $\subseteq$ **PMC$_{\text{CPMR}}$**.

4.2. Simulation of CPMRs by Turing machine in a polynomial space

In this subsection, the simulation of CPMRs by Turing machines in polynomial space is provided. Without loss of generality, we assume that the simulated P systems are confluent. In order to simulate non-deterministic confluent P systems, we define a weak priority between the selected rules, which is described as follows: (1) rules are used in a bottom-up manner for membranes; (2) rules are used in the order of types from (1) to (6) in a membrane [57,58]. In this way, confluent P systems can be simulated in a deterministic way.

The objects in any membrane h at any configuration $\mathcal{C}_t$ can be calculated from the previous configuration of this membrane.

We assume that the labels of the initial membranes are one-to-one, but in CPMRs, the membranes generated by a division rule have the same label. In order to identify the uniqueness of membranes, we need to relabel the membranes. Specifically, an index is added to each label of membranes in the following way:

1. The environment is a special membrane and the depth level of the environment is denoted by -1. The depth of the skin membrane is denoted as level 0. For the other membranes, the depth level of a membrane is the distance between the two nodes corresponding to this membrane and the skin membrane in the tree associated with the structure of the P system (see Fig. 2).
2. The index of the label of a membrane is the subscript of the label. In the initial configuration, each label has an empty index, that is, it has the form a, b, c, d, instead of $b_{[\dots]}$, $c_{[\dots,\dots]}$, etc., as shown in Fig. 2. The index of skin membrane remains unchanged during the computation process. In a configuration $\mathcal{C}_t$, the index of the label of a membrane with the depth level $k \geq 1$ is a k-tuple of the form $[s_1, \dots, s_k]$, where s_i, $1 \leq i \leq k$, is a string over 1 and 2, the length of a string s_i is the number of computation steps t, and the first $k-1$ strings are the same ones as the parent membrane. For example, in Fig. 2, after the first computation step, the label index of the membrane c with the depth level 2 is a

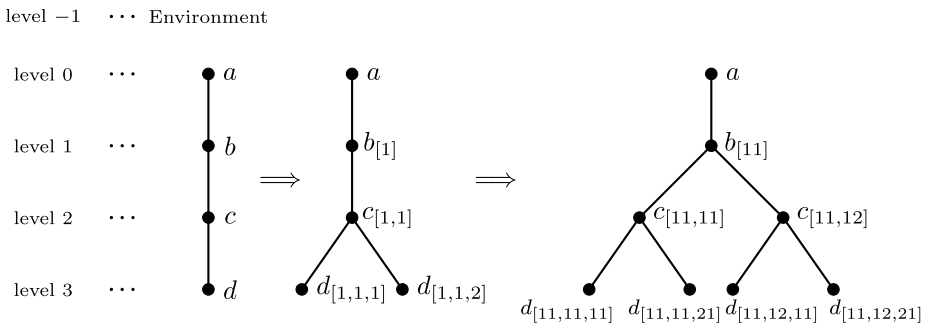

Fig. 2. The illustration of the updating of label indices of membranes during a computation, where the CPMR Π has the membrane structure $[\,[\,[\,[\,]_d\,]_c\,]_b\,]_a$. At step one, membrane with label d is divided; at step two, the non-elementary membrane $c_{1,1}$ is divided.

2-tuple $[1, 1]$, consisting of the two strings of length one 1 and 1, and the first string 1 is the same as the label index of the parent membrane b. The label indices of the membranes d with depth level 3 are 3-tuples [1,1,1] and [1,1,2], the first two strings of these two 3-tuples 1 and 1 are the same as the label index of the parent membrane c.

3. At a computation step, the label indices of membranes are extended in a top-down manner by the distances of nodes to the root node (corresponding to the skin membrane) in the tree structure of the P system, first root node, finally leaf nodes. Specifically, considering a membrane $h_{[i_{11}\ldots i_{1(t-1)},\ldots,i_{k1}\ldots i_{k(t-1)}]}$ at step $t-1$, if membrane h is not divided at step t, then after step t the first $k-1$ strings of the label index of membrane h are the same as the parent membrane and the digit 1 is added to the last string $i_{k1}\ldots i_{k(t-1)}$ in the right end, that is, $h_{[i_{11}\ldots i_{1(t-1)},\ldots,i_{k1}\ldots i_{k(t-1)}]}$ is updated to $h_{[s_1,\ldots,s_{k-1},i_{k1}\ldots i_{k(t-1)}1]}$, where $s_1,\ldots,s_{k-1}$ are the label index of the parent membrane (which is already updated following the top-down manner). If membrane h is divided at step t, then after step t the first $k-1$ strings of the label index of membrane h are the same as the parent membrane, and digits 1 and 2 are added to the last string $i_{k1}\ldots i_{k(t-1)}$ in the right end, respectively; the obtained two k-tuples in this way are the updated label indices of the two new generated membranes, correspondingly. For example, in Fig. 2, at step 2, membrane with label b is not divided, so the label index of this membrane is updated from [1] to [11] by adding digit 1 to the last string (the only string) 1, that is, the label $b_{[1]}$ is updated to $b_{[11]}$. The membrane with label c is divided at step 2, two new membranes with label c are generated, the label indices of membranes c are 2-tuples as the depth level of these membranes is 2, the first string of these 2-tuples are 11, which is the same as the parent membrane b, the last string 1 of the tuple $[1, 1]$ is updated to $[11, 12]$ by adding 1 and 2 in the right end, respectively. Now, the labels of membrane c become $c_{[11,11]}$ and $c_{[11,12]}$.

Consider a CPMR $\Pi = (\Gamma, H, \Sigma, \mu, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_m, \mathcal{R}, i_{in}, i_{out})$. We denote by $S.M$ the multiset of objects in membrane M (S is the *state* of each membrane). If a membrane does not exist, then the state of such membrane is denoted by *null*. We denote by $|S.M|_a$ the multiplicity of an object a in $S.M$.

In what follows, we define Function `State`, which is used to decide the result of a computation of Π, specifically, from the initial configuration to a halting configuration. Function `State` can compute any membrane h which potentially exists at any step, and remember whether a rule was used in the process of computation. Function `Parent` and Function `Use_children` are used to calculate the parent membrane and the multisets of objects in membranes, respectively. Function `State` is used to calculate the state of the parent membrane and change the content of a membrane when rules are used in its children membranes, respectively. Functions `Use_rules_type(1)-Use_rules_type(6)` are used to calculate the content of membrane after applying the corresponding types of rules.

Function `State`

Input:

$h_{[i_{11}\ldots i_{kt}]}$ (a specification of a membrane), $t \geq 0$: a nonnegative integer, $k \geq 0$ is the depth level of membrane.

Variables:

S, T: states of membrane $h_{[i_{11}\ldots i_{k(t-1)}]}$ and its parent membrane at configuration $\mathcal{C}_{t-1}$, respectively;

T': state of parent membrane of $h_{[i_{11}\ldots i_{kt}]}$ at configuration $\mathcal{C}_t$.

Output:

S': state of membrane $h_{[i_{11}\ldots i_{kt}]}$ at configuration $\mathcal{C}_t$.

1. If $t = 0$, then output the state of membrane h in the initial configuration $\mathcal{C}_0$ and the program exits.
2. If $k \geq 0$, then
   (a) $T \leftarrow$ `State`$(\texttt{Parent}(h_{[i_{11}\ldots i_{kt}]}), t)$
   (b) If $T =$ *null*, then *null* is returned and the program exits.
3. $S \leftarrow$ `State`$(h_{[i_{11}\ldots i_{k(t-1)}]}, t-1)$
4. If $S =$ *null*, then *null* is returned and the program exits.
5. $T' \leftarrow \emptyset$, $S' \leftarrow \emptyset$
6. `Use_children`$(h_{[i_{11}\ldots i_{kt}]}, t, S.M, S'.M)$

7. If $k \geq 0$, then
   (a) $T \leftarrow$ `State(Parent(`$h_{[i_{11}\ldots i_{k(t-1)}]}$`)`$, t-1)$
   (b) `Use_rules_type(1)` $(h, S, S', T.M, T'.M)$
   (c) `Use_rules_type(2)` $(h, S, S', T.M, T'.M)$
   (d) `Use_rules_type(3)` $(h, S, S', T.M, T'.M)$
   (e) `Use_rules_type(4)` $(h, S, S', T.M, T'.M)$
   (f) `Use_rules_type(5)` $(h, S, S', T.M, T'.M)$
   (g) `Use_rules_type(6)` (h, S, S', i_{kt})
8. If $S' \neq null$, then $S'.M \leftarrow S'.M \cup S.M$.
   /* All unused objects pass unchanged to the next configuration. */
9. If $i_{kt} = 2$ (an index 2 is added to a label of membrane, which means such membrane is obtained by a division rule) and rule of type (6) cannot be used, then $S' \leftarrow null$. /* Membrane $h_{[i_{11}\ldots i_{kt}]}$ was possibly created by an application of a rule of type 6 in t-th step. If the rule was not applied, then membrane $h_{[i_{11}\ldots i_{kt}]}$ does not exist. */
10. Output S' and exit.

Function `Parent`
Input:
$h_{[i_{11}\ldots i_{kt}]}$: a specification of a membrane.
Output:
At configuration $\mathcal{C}_t$, the parent membrane of $h_{[i_{11}\ldots i_{kt}]}$.
1. Return $g_{[i_{11}\ldots i_{(k-1)t}]}$. /* g is the label of the parent membrane of h in the initial configuration. */

As rules in the CPMR are used with a bottom-up priority, objects are evolved through elementary membranes to their parent membranes, which will change the contents of these membranes, and influence rules that are used in these membranes.

Function `Use_children`
Input:
$h_{[i_{11}\ldots i_{kt}]}$: a specification of a membrane;
$t \geq 0$: a nonnegative integer;
M_1: multiset of objects in membrane $h_{[i_{11}\ldots i_{kt}]}$ at configuration $\mathcal{C}_t$;
M_2: empty set.
Variables:
T, T': states of membranes.
Output:
$M_1 \leftarrow M_1$ minus the multiset of objects, which are removed from membrane $h_{[i_{11}\ldots i_{kt}]}$ at the next step;
$M_2 \leftarrow M_2$ plus the multiset of objects, which come from other membranes at the next step.

For each t-tuple $[j_{(k+1)1}\ldots j_{(k+1)t}]$ ($j_{(k+1)l} \in \{1, 2\}$, $1 \leq l \leq t$) and each children membrane g of h, do:

1. $T \leftarrow$ `State(`$g_{[i_{11}\ldots i_{k(t-1)}, j_{(k+1)1}\ldots j_{(k+1)(t-1)}]}, t-1$`)`, $T' \leftarrow \emptyset$
2. If $T = null$, then end this cycle.
3. `Use_children(`$g_{[i_{11}\ldots i_{kt}, j_{(k+1)1}\ldots j_{(k+1)t}]}, t, T.M, T'.M$`)`
4. `Use_rules_type(1)` (g, T, T', M_1, M_2)
5. `Use_rules_type(2)` (g, T, T', M_1, M_2)
6. `Use_rules_type(3)` (g, T, T', M_1, M_2)
7. `Use_rules_type(4)` (g, T, T', M_1, M_2)
8. `Use_rules_type(5)` (g, T, T', M_1, M_2)

Finally, the following functions evaluate a simultaneous application of rules in a specified membrane.

Functions `Use_rules_type(1)-Use_rules_type(6)`
Input:
h: a specification of membrane;
S, S': two states of the membrane;
M_1, M_2: two multisets of objects in the parent membrane of h (for rules of types (1)-(5)); i: index of a membrane (for rules of type (6)).
Output:
$S \leftarrow S$ minus multiset of objects consumed in membrane h by the applied rule
$S' \leftarrow S'$ plus multiset of objects produced in membrane h by the applied rule
$M_1 \leftarrow M_1$ minus multiset of objects consumed in the parent membrane by the applied rule
$M_2 \leftarrow M_2$ plus multiset of objects produced in the parent membrane by the applied rule

Here we demonstrate only the functions for rules of types (5) and (6). Function for rules of types (1)-(4) are analogous to rules of type (5).

(5) For each rule $a[\ b\ ]_i^{\alpha_1} \to c[\ d\ ]_i^{\alpha_2}$ in R_h:

- remove a copy of a from M_1
- remove a copy of b from $S.M$
- add a copy of c to M_2
- add a copy of d to $S'.M$

(6) For each rule $[\ a\ ]_i^{\alpha_1} \to [\ b\ ]_i^{\alpha_2}[\ c\ ]_i^{\alpha_3}$ in R_h:

if $a \in S.M$ then

- remove a from $S.M$
- if $i = 1$ then add b to $S'.M$ else add c to $S'.M$
- skip all other rules of type (6)

Note that in a CPMR, only one object and only one membrane can be subject to one rule of any type. Hence any two (or more than two) types of rules referring to the same membrane cannot be used at the same step.

4.2.1. Formal verification of the simulation

Theorem 4.2. *Let Π be a confluent recognizer CPMR and let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation with weak priority in rules as mentioned above. Let $h_{[i_{11}\dots i_{kt}]}$ be a membrane of Π at a configuration $\mathcal{C}_t$ in $\mathcal{C}$. We can obtain the following statements: if at the configuration $\mathcal{C}_t$ the system Π has membrane $h_{[i_{11}\dots i_{kt}]}$, then the state of membrane h is obtained by function* `State`$(h_{[i_{11}\dots i_{kt}]}, t)$*; otherwise, null is obtained from the function* `State`$(h_{[i_{11}\dots i_{kt}]}, t)$.

Proof. It is easy to check that functions `Parent` and `Use_rules_type(x)` $(1 \le x \le 6)$ are independent of any other function. By induction on t and k, we prove that the recursive functions `State` and `Use_children` correctly output the multisets of objects as the system Π works.

1. Let $t = 0$. The function `State` returns the state of a membrane at the configuration $\mathcal{C}_0$ and so the result holds.
2. By induction hypothesis, let the theorem hold for $t \ge 0$. We will prove that it holds also for $t + 1$.
   (a) Let h be the environment, so $k = -1$. In this case, we analyse each item in function `State`.
      i. Item 2 cannot be used.
      ii. Item 3 outputs the correct state of membrane h according to induction hypothesis.
      iii. Item 4 cannot be used because the environment exists at any time.
      iv. By induction hypothesis, function `Use_children` with parameter t will output the correct result.
      v. Item 7 cannot be used.
      vi. Item 8 cannot be used due to the empty index for the environment.
      vii. Item 9, objects which are not involved by any rule will be introduced into the resulting membrane.
   (b) By induction hypothesis, let function `State` output the correct results for $k \ge -1$. We will prove that this property holds also for $k + 1$.
      i. For item 2, if $k \ge 0$, then function `State` is recursively called. Besides, k is the depth level of its parent membrane, hence by induction hypothesis, the result is correct.
      ii. Item 3 outputs the correct state of membrane h according to induction hypothesis.
      iii. For item 4, if at configuration $\mathcal{C}_t$, the membrane h does not exist, then function `State` outputs *null*.
      iv. For item 6, by induction hypothesis, function `Use_children` with parameter t is used, hence it returns correct result.
      v. For item 7, recursively call function `State` with parameter t, which outputs the correct result by induction hypothesis.
      vi. Item 8, if the system uses a rule of type (6), the number 2 is the last digit of the index of the membrane; otherwise this index of the membrane does not exist, and *null* must be returned.
      vii. Item 9, objects which are not involved by any rule will be introduced into the state of the membrane.
3. We will prove that the function `Use_children` correctly outputs the multisets $M_2, S'.M$ of objects. Obviously, for item 6 of Function `State`, function `Use_children` with parameters $h_{[i_{11}\dots i_{kt}]}$ and t is used in function `State` with parameter $t + 1$. The depth of the membrane system is denoted by d.
   (a) If $k = d$, then we know that membrane $h_{[i_{11}\dots i_{kt}]}$ is an elementary membrane, hence function `Use_children` in the main cycle cannot be used, and the multisets M_1 and M_2 are returned unchanged.
   (b) For a depth level k, if the function `Use_children` correctly outputs the multisets of objects, then it holds also for depth level $k - 1$.

i. The call to the function `State` with parameter t for item 1, according to induction hypothesis for t, returns the correct result.
ii. The recursive call to the function `Use_children` with parameters k and t in item 3, by induction hypothesis on k, returns the correct result.
iii. Rules in membranes of $h_{[i_{11}\ldots i_{kt}]}$ are simulated in items 4-8, which are correct by assumption.

Therefore, for any step t and membrane with the depth level k, the function `State` with parameters $(h_{[i_{11}\ldots i_{kt}]}, t)$ correctly outputs the state of membrane $h_{[i_{11}\ldots i_{kt}]}$. □

Corollary 4.2. *Let Π be a confluent CPMR. The computation result of Π can be obtained by function* `State`.

Proof. The output membrane of Π is the skin membrane, denoted by i_0. It is known that object `yes` or `no` appears in i_{out} at the last step of the computation, which can be calculated by function `State(`i_{out}, t`)` ($t \geq 0$). Considering Theorem 4.2 and the definition of confluent P systems, we can conclude that the system outputs the correct results. □

4.2.2. Space complexity of the simulation

We denote by $\mathbf{MC_{CPMR}}(t(n))$ the class of problems solvable by a sound and complete polynomially uniform family of recognizer CPMR in time $t(n)$.

Theorem 4.3. *Let $t(n) \geq n$, then we have $\mathbf{MC_{CPMR}}(t(n)) \subseteq SPACE(t(n)^{O(1)})$.*

Proof. Let $w \in I_X$ be an instance with size $n = |w|$ of a decision problem X, which is solved by a sound and complete uniform family of recognizer CPMRs. According to Corollary 4.2, it amounts to determine the space complexity of the function `State` to prove the statement.

Let the skin membrane be labeled with 1 and we denote by d the depth of the membrane structure; the number of objects initially placed in the system is denoted by q; the number of objects at configuration $\mathcal{C}_t$ is denoted by o_t, hence $o_0 = |\mathcal{M}_1| + \cdots + |\mathcal{M}_m|$.

It is easy to see that the values of m, d, q and $\log o_0$ are bounded by $n^{\mathcal{O}(1)}$. We treat these values as constants as they are fixed for a given system $\Pi(w)$ in the rest of proof. Moreover, the number of objects in the system can increase in the following two cases: (i) by introducing objects from the environment into the system, hence at configuration $\mathcal{C}_t$, at most t copies of objects can be inserted; (ii) by membrane division, at configuration $\mathcal{C}_t$, the number of membranes is bounded by $m(2^d)^t$. Hence at configuration $\mathcal{C}_t$ the total number of objects $o_t \leq (o_0 + t)m(2^d)^t$.

Since all objects in the system can appear in one membrane, hence at configuration $\mathcal{C}_t$, in order to store the contents of an arbitrary membrane at configuration $\mathcal{C}_t$, we need the space (in bits)

$$b_t \leq q\lceil \log o_t \rceil \leq q\lceil \log (o_0 + t)m \rceil + qdt \leq c_0 + c_1 t, \tag{1}$$

for positive constants $c_0, c_1 \in n^{\mathcal{O}(1)}$.

Let us denote by $S(t,k)$ and $C(t,k)$ the space complexities of function `State` with the parameter $h_{[i_{11}\ldots i_{kt}]}$ and of function `Use_children`, respectively. According to Function `State`, we know that `State(t,k)` calls `State(t,k-1)`, `State(t-1,k)`, `Use_children(t,k)` and `State(t-1,k-1)`. From Function `Use_children`, we know that `Use_children` calls `State(t-1,k+1)` and `Use_children(t,k+1)`; besides, the content of membrane $h_{[i_{11}\ldots i_{kt}]}$ (and its parent/child) is stored by variables as S, S', T, T', and each of them requires at most b_t bits; a specification of membrane $h_{[i_{11}\ldots i_{kt}]}$ requires $kt + c$ (we denote by $c = \lceil \log m \rceil$) bits. Let

$$S(t) = \max\{S(t,k) \mid 0 \leq k \leq d\}. \tag{2}$$

The following recurrences are obtained:

$$S(0,k) = b_0, 0 \leq k \leq d; \tag{3}$$

$$S(t,0) = \max\{C(t,0), S(t-1,0)\} + 4b_t + c, t \geq 1; \tag{4}$$

$$\begin{aligned} S(t,k) = {} & \max\{C(t,k), S(t,k-1), S(t-1,k), S(t-1,k-1)\} + 4b_t + kt + c, \\ & t \geq 1, 1 \leq k \leq d; \end{aligned} \tag{5}$$

$$C(t,d) = 0; \tag{6}$$

$$\begin{aligned} C(t,k) \leq {} & \max\{C(t,k+1), S(t-1,k+1)\} + 4b_t + t(k+1) + c, \\ & t \geq 0, 0 \leq k < d. \end{aligned} \tag{7}$$

By expanding (7) for $k, k+1, \ldots, d$, respectively, we obtain

$$C(t,k) \leq \max\{S(t-1,i) \mid k < i \leq d\} + \mathcal{O}(d^2t + db_t), 0 \leq k \leq d, t \geq 0. \tag{8}$$

From (2) and (8), formula (5) can be rewritten as follows:

$$S(t,k) \leq \max\{S(t-1) + \mathcal{O}(d^2t + db_t)\} + O(b_t + kt), t \geq 1, 1 \leq k \leq d. \tag{9}$$

By substituting $S(t,i)$ $(0 \leq i \leq k-1)$ with formula (9), we can obtain

$$\begin{aligned} S(t,k) &\leq \max\{S(t,k-1), S(t-1) + O(d^2t + db_t)\} + 2O(b_t + kt) \\ &\leq \max\{S(t,k-2), S(t-1) + O(d^2t + db_t)\} + 3O(b_t + kt) \\ &\cdots \\ &\leq \max\{S(t,0), S(t-1) + O(d^2t + db_t)\} + (k+1)O(b_t + kt) \\ &\leq \max\{C(t,0) + 4b_t + c, S(t-1) + O(d^2t + db_t)\} + O(kb_t + k^2t) \\ &\leq S(t-1) + O(d^2t + db_t). \end{aligned}$$

Therefore, according to (2), the recurrence (3)-(5) can be rewritten as follows:

$$S(0) = b_0; \tag{10}$$

$$S(t) \leq S(t-1) + \mathcal{O}(d^2t + db_t). \tag{11}$$

From (11), we can obtain

$$S(t) = \mathcal{O}(d^2t^2 + tdb_t). \tag{12}$$

Recall that $d = n^{\mathcal{O}(1)}$ (n is the original instance size). From (1), we obtain $S(t) = (nt)^{\mathcal{O}(1)}$. Finally, we assume that the computation of the CPMR is polynomial time bounded, denoted by $t(n)$, then we obtain $S(t(n)) = (nt(n))^{\mathcal{O}(1)} = t(n)^{O(1)}$ (by assumption $t(n) \geq n$). □

So, the following result is obtained.

Corollary 4.3. $\mathbf{PMC_{CPMR}} \subseteq \mathbf{PSPACE}$.

From Corollaries 4.1 and 4.3, we obtain the following result:

Theorem 4.4. $\mathbf{PMC_{CPMR}} = \mathbf{PSPACE}$.

5. Conclusions and further works

In this work, we considered a normal form for P systems with active membranes, called cell-like P systems with polarizations and minimal rules (CPPMRs, for short). The computation power of CPPMRs has been investigated. We have shown that CPPMRs with arbitrarily many membranes, with three polarizations, using rules of types (1), (2) or using rules of types (3), (5) or using only rules of type (4) are able to compute only finite sets of non-negative integers. Besides, we proved that CPPMRs with four membranes, with three polarizations, using rules of types (1), (2), (4) or using rules of types (3), (4) are Turing universal; the result about universality can also be obtained by CPPMRs with two membranes, two polarizations and using rules of types (4), (5). Moreover, we have shown that the class of problems that can be solved in polynomial time by P systems with minimal rules and without polarizations is equal to the class of problems **PSPACE**.

The environment plays an important role for the computation power in CPPMRs, since the number of objects in the systems can be increased (using rules of type (4)) if and only if the objects sent into the systems are initially placed in the environment. Obviously, if the alphabet of the environment of CPPMRs is empty, then CPPMRs can only generate finite sets of natural numbers.

In the area of membrane computing, there exist three methods to produce new membranes, all of which are inspired by living cells: membrane division (mitosis) [37], membrane separation (membrane fission) [29], and membrane creation (autopoiesis) [26]. The computation model CPPMR considered in this work has the rule of membrane division. It is of interest to investigate cell-like P systems with polarizations and minimal rules where membrane division is replaced with membrane separation or membrane creation.

The **P** versus **NP** problem is a major unsolved problem in theoretical computer science. In computational complexity theory, the methods to tackle **P** versus **NP** problem were presented in terms of syntactical or semantical ingredients of the models, and many interesting characterizations of the $\mathbf{P} \neq \mathbf{NP}$ conjecture arise in membrane computing [13,20,31,53]. It is interesting to investigate the **P** versus **NP** problem in the framework of CPPMRs.

Declaration of competing interest

The authors declare that they have no conflicts of interest.

Acknowledgements

The work was supported by National Natural Science Foundation of China (61972138, 61602192, 61320106005 and 61772214), the Fundamental Research Funds for the Central Universities (531118010355). The work of the second and fourth authors are also supported by the research project TIN2017-89842-P, cofinanced by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

[1] A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes, Fundam. Inform. 58 (2003) 67–77.
[2] A. Alhazov, L. Pan, Polarizationless P systems with active membranes, Grammars 7 (2004) 141–159.
[3] A. Alhazov, M.J. Pérez-Jiménez, Uniform solution of QSAT using polarizationless active membranes, Lect. Notes Comput. Sci. 4664 (2007) 122–133.
[4] M. Cavaliere, D. Sburlan, Time–independent P systems, Lect. Notes Comput. Sci. 3365 (2005) 239–258.
[5] G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez, P systems with minimal parallelism, Theor. Comput. Sci. 378 (2007) 117–130.
[6] J. Cooper, R. Nicolescu, Alternative representations of P systems solutions to the graph colouring problem, J. Membr. Comput. 1 (2) (2019) 112–126.
[7] R. Freund, M. Oswald, GP systems with forbidding context, Fundam. Inform. 49 (1) (2002) 81–102.
[8] R. Freund, A. Păun, P systems with active membranes and without polarizations, Soft Comput. 9 (2005) 657–663.
[9] R. Freund, S. Verlan, (Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode, Nat. Comput. 10 (2) (2010) 821–833.
[10] P. Frisco, G. Govan, P systems with active membranes operating under minimal parallelism, Lect. Notes Comput. Sci. 7184 (2012) 165–181.
[11] P. Frisco, G. Govan, A. Leporati, Asynchronous P systems with active membranes, Theor. Comput. Sci. 429 (2012) 74–86.
[12] P. Frisco, H.J. Hoogeboom, P systems with symport/antiport simulating counter automata, Acta Inform. 41 (2) (2004) 145–170.
[13] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero, Computational efficiency of dissolution rules in membrane systems, Int. J. Comput. Math. 83 (7) (2006) 593–611.
[14] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2–3) (2006) 279–308.
[15] S.N. Krishna, R. Rama, A variant of P systems with active membranes: solving NP–complete problems, Rom. J. Inf. Sci. Technol. 2 (1999) 357–367.
[16] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Constant-space P systems with active membranes, Fundam. Inform. 134 (1–2) (2014) 111–128.
[17] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, A survey on space complexity of P systems with active membranes, Int. J. Adv. Eng. Sci. Appl. Math. 10 (3) (2018) 221–229.
[18] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Subroutines in P systems and closure properties of their complexity classes, Theor. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2018.06.012, in press.
[19] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Characterizing PSPACE with shallow non-confluent P systems, J. Membr. Comput. 1 (2) (2019) 75–84.
[20] L.F. Macías-Ramos, B. Song, L. Valencia-Cabrera, L. Pan, M.J. Pérez-Jiménez, Membrane fission: a computational complexity perspective, Complexity 21 (6) (2016) 321–334.
[21] L.F. Macías-Ramos, L. Valencia-Cabrera, B. Song, L. Pan, M.J. Pérez-Jiménez, A P-Lingua based simulator for P systems with symport/antiport rules, Fundam. Inform. 139 (2) (2015) 211–277.
[22] A. Maroosi, R.C. Muniyandi, Accelerated execution of P systems with active membranes to solve the N-queens problem, Theor. Comput. Sci. 551 (2014) 39–54.
[23] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón, Theor. Comput. Sci. 296 (2) (2003) 295–326.
[24] M.A. Martínez-del-Amor, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, Simulating P systems on GPU devices: a survey, Fundam. Inform. 136 (3) (2015) 269–284.
[25] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.
[26] M. Mutyam, K. Krithivasan, P systems with membrane creation: universality and efficiency, Lect. Notes Comput. Sci. 2055 (2001) 276–287.
[27] D. Orellana-Martín, M.A. Martínez-del-Amor, I. Pérez-Hurtado, A. Riscos-Núñez, L. Valencia-Cabrera, M.J. Pérez-Jiménez, When object production tunes the efficiency of membrane systems, Theor. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.04.013, in press.
[28] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems, J. Membr. Comput. 1 (2) (2019) 85–92.
[29] L. Pan, T.-O. Ishdorj, P systems with active membranes and separation rules, J. Univers. Comput. Sci. 10 (5) (2004) 630–649.
[30] L. Pan, Gh. Păun, B. Song, Flat maximal parallelism in P systems with promoters, Theor. Comput. Sci. 623 (2016) 83–91.
[31] L. Pan, M.J. Pérez-Jiménez, Computational complexity of tissue-like P systems, J. Complex. 26 (3) (2010) 296–315.
[32] L. Pan, B. Song, L. Valencia-Cabrera, M.J. Pérez-Jiménez, The computational complexity of tissue P systems with evolutional symport/antiport rules, Complexity (2018) 3745210, 21 pages.
[33] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, Reading, Mass, 1994.
[34] A. Păun, On P systems with active membranes, in: Proceedings of the Second International Conference on Unconventional Models of Computation, (UMC'2K), 2000, pp. 187–201.
[35] A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, New Gener. Comput. 20 (3) (2002) 295–305.
[36] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[37] Gh. Păun, P systems with active membranes: attacking NP-complete problems, J. Autom. Lang. Comb. 6 (1) (2001) 75–90.
[38] Gh. Păun, Computing with Membranes: An Introduction, Springer-Verlag, Berlin, 2002.
[39] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, New York, 2010.
[40] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, Inf. Sci. 304 (2015) 80–91.
[41] H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, Inf. Sci. 235 (2013) 106–116.

[42] M.J. Pérez-Jiménez, An approach to computational complexity in membrane computing, Lect. Notes Comput. Sci. 3365 (2005) 89–109.
[43] M.J. Pérez-Jiménez, A. Riscos-Núñez, Solving the Subset-Sum problem by active membranes, New Gener. Comput. 23 (2005) 367–384.
[44] M.J. Pérez-Jiménez, P. Sosík, An optimal frontier of the efficiency of tissue P systems with cell separation, Fundam. Inform. 138 (1–2) (2015) 45–60.
[45] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, 3 vols, Springer, Berlin, 1997.
[46] T. Song, L. Pan, Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy, IEEE Trans. Nanobiosci. 14 (2015) 38–44.
[47] T. Song, L. Pan, Spiking neural P systems with rules on synapses working in maximum spiking strategy, IEEE Trans. Nanobiosci. 14 (4) (2015) 465–477.
[48] T. Song, L. Pan, Spiking neural P systems with request rules, Neurocomputing 193 (2016) 193–200.
[49] B. Song, Y. Kong, Solution to PSPACE-complete problem using P systems with active membranes with time-freeness, Math. Probl. Eng. (2019) 5793234.
[50] B. Song, M.J. Pérez-Jiménez, L. Pan, Efficient solutions to hard computational problems by P systems with symport/antiport rules and membrane division, Biosystems 130 (2015) 51–58.
[51] B. Song, M.J. Pérez-Jiménez, Gh. Păun, L. Pan, Tissue P systems with channel states working in the flat maximally parallel way, IEEE Trans. Nanobiosci. 15 (7) (2016) 645–656.
[52] B. Song, T. Song, L. Pan, A time-free uniform solution to subset sum problem by tissue P systems with cell division, Math. Struct. Comput. Sci. 27 (1) (2017) 17–32.
[53] B. Song, C. Zhang, L. Pan, Tissue-like P systems with evolutional symport/antiport rules, Inf. Sci. 378 (2017) 177–193.
[54] P. Sosík, The computational power of cell division in P systems: beating down parallel computers?, Nat. Comput. 2 (3) (2003) 287–298.
[55] P. Sosík, L. Cienciala, Computational power of cell separation in tissue P systems, Inf. Sci. 279 (2014) 805–815.
[56] P. Sosík, L. Cienciala, A limitation of cell division in tissue P systems by PSPACE, J. Comput. Syst. Sci. 81 (2015) 473–484.
[57] P. Sosík, A. Păun, A. Rodríguez-Patón, P systems with proteins on membranes characterize PSPACE, Theor. Comput. Sci. 488 (2013) 78–95.
[58] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: a characterization of PSPACE, J. Comput. Syst. Sci. 73 (2007) 137–152.
[59] L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez–Jiménez, Cooperation in transport of chemical substances: a complexity approach within membrane computing, Fundam. Inform. 154 (2017) 373–385.
[60] L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez–Jiménez, Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation, Theor. Comput. Sci. 701 (2017) 226–234.
[61] S. Verlan, J. Quirós, Fast hardware implementations of P systems, Lect. Notes Comput. Sci. 7762 (2013) 404–423.
[62] T. Wu, Z. Zhang, Gh. Păun, L. Pan, Cell-like spiking neural P systems, Theor. Comput. Sci. 623 (2016) 180–189.
[63] C. Zandron, A. Leporati, C. Ferretti, G. Mauri, M.J. Pérez–Jiménez, On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution, Fundam. Inform. 87 (1) (2008) 79–91.
[64] G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: a comprehensive survey and new results, Inf. Sci. 279 (2014) 528–551.