An efficient time-free solution to QSAT problem using P systems with proteins on membranes

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

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

[b] School of Electric and Information Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, China

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

Keywords:
Bio-inspired computing
Membrane computing
P system
Time-free solution
QSAT problem

A B S T R A C T

P systems are a class of distributed parallel computing devices inspired by some basic behaviors of biological membranes, which have the restriction that each rule is executed in exactly one time unit. However, it is natural to consider the systems without the time restriction on each rule since biochemical reactions in biological systems are inherently parallel and have different reaction rates, and the execution time of biochemical reactions is unpredictably sensitive to environmental factors. In this work, we construct a family of P systems with proteins on membranes and membrane division that are "robust" against the execution time of rules. Specifically, we present a time-free uniform solution to the QSAT problem by using P systems with proteins on membranes and membrane division in the sense that the execution time of the involved rules has no influence on the correctness of the solution.

1. Introduction

Membrane computing is a naturally inspired computational paradigm, abstracting computational ideas (e.g., computational models, data structures, data operations) from the structure and the functioning of living cells. The aim of membrane computing is to produce a family of coherent, robust and efficient computational models. This direction of research area was initiated in 1998 and it has developed quickly on both theoretical results [2,37,39,40,42] and application of solving real problems [29,30,43,44]. All classes of computational models considered in the framework of membrane computing are called *P systems*, which are parallel and distributed computational models. Three main types of P systems have been considered until now from the point of biological motivation: cell-like P systems (inspired by the living cell with its compartments arranged in a hierarchical structure) [26], tissue-like P systems (inspired by cell inter-communication in tissues and in certain environment) [15], neural-like P systems (inspired by the way the neurons communicate by means of electrical impulses of identical shape) [12]. One may refer to the handbook of membrane computing for general information [28], and the most up-to-date information for this area, please refer to the P systems webpage: http://ppage.psystems.eu. In this work, we deal with a class of cell-like P systems: *P systems with objects on membranes*.

* Corresponding author at: Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China.

E-mail addresses: boshengsong@hust.edu.cn (B. Song), marper@us.es (M.J. Pérez-Jiménez), lqpan@mail.hust.edu.cn (L. Pan).

P systems with objects (also called *proteins*) on membranes are motivated by the biological reality that some biochemical reactions taking place in a cell involve proteins bound on membranes such as transportation of substrates and even are regulated by proteins on membranes [1]. Membrane proteins can be divided into two major types with respect to the way they are associated to the lipid bilayer: *peripheral* proteins, placed on one side of a membrane, internal or external, and *integral* (also called *transmembrane*) proteins, which have parts of the molecule on both sides of the membrane (it is estimated that in the animal cells, the proteins constitute about half of the mass of the membranes [1]). In general, there are three main types of P systems with objects on membranes investigated: (1) (mem)brane systems (inspired by brane calculi), objects evolve together with membranes by operations of pino, exo, phago calculus and mate, drip, bud calculus [4,5,27]; (2) Ruston models, objects placed on membranes control the evolution of objects in the neighboring regions [13,22–25]; (3) Trento models, objects are placed both on membranes and in regions, and they can change their places [3,7–9].

The present work deals with P systems with proteins on membranes, specifically, Ruston models, which are the models combining membrane systems and brane calculi as introduced in [23]. A P system with proteins on membranes consists of a hierarchical arrangement of membranes, each membrane delimiting a compartment (also called *region*). A membrane with no compartments inside is called *elementary*, otherwise it is called *non-elementary*. The outmost membrane is called a *skin* membrane, the space outside the skin membrane is called the *environment*. In such P systems, we have objects both **in** compartments, as usual in P systems, and **on** membranes (called *proteins*), the proteins are fixed to the membranes and used mainly to control the evolution of other objects. The evolution rules used in P systems with proteins on membranes are inspired from the symport/antiport rules (symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions) [20], but the objects can evolve under the control of proteins placed on membranes (the proteins can also change). Moreover, P systems with proteins on membranes always use minimal rules, that is, dealing with only one protein, one object inside the region and/or one object outside of it.

The computational power of P systems with proteins on membranes was investigated in [23], where many universal results of such P systems are obtained by combining several various types of rules. A particular class of P systems with proteins on membranes, called *flip-flop membrane systems with proteins*, was also investigated in [13,25], where each protein has at most two states, and flip-flop membrane systems with proteins are proved to be universal by simulating register machines. If membrane division is introduced in P systems with proteins on membranes, computationally hard problems can be solved in a feasible time by a time–space trade-off. In [24], the SAT problem was solved by a uniform family of P systems with proteins on membranes and membrane division in polynomial time. P systems with proteins on membranes were also used to solve **PSPACE**-complete problems. In [41], an efficient solution to the QSAT problem was given by a family of recognizer P systems with proteins on membranes in a semi-uniform way, moreover, it was shown that the polynomial complexity class (semi-uniform version) associated with recognizer P systems with proteins on membranes coincides with the class of **PSPACE**-complete problems. All the above mentioned P systems with proteins on membranes work in a synchronized and parallel way, that is, there exists a global clock which marks the time for the whole system, in each time unit, all the rules evolve synchronously and the execution time of each rule takes exactly one time unit (one step). From a biological point of view, a more realistic way is to consider the system without the time restriction on each rule, because the execution time of biochemical reactions is typically exponentially distributed, at least for well-mixed mass-action systems, and it is sensitive to environmental factors that could affect reaction rates in an unpredictable way.

With this biological motivation, P systems that use time as support for the computation are proposed (called *timed P systems*), in such P systems, a natural number (represented the execution time of the rule) is associated to each rule [6]. A particular class of timed P systems, called *time-free P systems*, was also formulated in [6], by simulating non-synchronized P systems, time-free P systems were proved Turing universal when using bi-stable catalysts and priority. Time-free solutions to computationally hard problems as open problems were formulated in [11]. The first attempt in this topic was done in [34], where a family of P systems with active membranes was designed for a time-free solution to the SAT problem in the sense that correctness of the solution does not depend on the execution time of the involved rules. Since then, several variants of P systems were used to solve **NP**-complete problems in a time-free manner [35,38]. However, it remains open whether **PSPACE**-complete problems can be solved by P systems with proteins on membranes in the time-free context.

In this work, we design a family of P systems with proteins on membranes and membrane division, which solves **PSPACE**-complete problem, the QSAT problem, in a time-free uniform manner in the sense that a P system constructed can solve a family of instances with the same size and correctness of the solution does not depend on the execution time of the involved rules.

2. P systems with proteins on membranes and membrane division

2.1. P systems with proteins on membranes and membrane division

The P systems considered in this work use two types of objects: *proteins* and simple *objects* [23]. The proteins are placed **on** the membranes, which never leave their places during the evolution process (but they can evolve) and are used mainly to control the evolution of other objects; the simple objects are placed **in** the regions delimited by membranes, which can leave from the regions to the neighboring regions during the evolution. If a protein p is placed on a membrane (with label) i and an object a is placed in the region delimited by that membrane, then it is denoted by $[\ p \mid a\ ]_i$ (note that if a membrane is divided by a division rule, then the new membranes resulting from division have the same label with the divided membrane; that is, duplicate labels are possible; of course, membranes with a same label are subject to the same set of rules). The regions of a membrane structure and the membranes themselves can contain multisets of objects and of proteins, respectively.

P systems with proteins on membranes and membrane division are used for solving decision problems, where an exponential workspace can be generated in polynomial or linear time, and computationally hard problems can be solved by trading off space for time (see [24,41]).

Before introducing the definition of P systems with proteins on membranes and membrane division, we first give some basic notions and notations from formal language theory [33].

An alphabet Σ is a finite and non-empty set of symbols. An ordered finite sequence of symbols form a *string* or *word*. A string over Σ is obtained by juxtaposing symbols of Σ. The empty string is denoted by λ. The set of all strings over an alphabet Σ is denoted by Σ^* and by $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ we denote the set of non-empty strings.

A *multiset* m over an alphabet Σ is a pair (Σ, f) where $f : \Sigma \to \mathbb{N}$ is a mapping, $\mathbb{N}$ is the set of non-negative integers. If $m = (\Sigma, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in \Sigma \mid f(x) > 0\}$. If $m = (\Sigma, f)$ is a finite multiset over Σ, and $supp(m) = \{a_1, \dots, a_k\}$, then it will be denoted as $m = \{a_1^{f(a_1)}, \dots, a_k^{f(a_k)}\}$. We usually represent m by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ or by any permutation of this string.

The union and relative complement of two multisets are defined as follows. Let $m_1 = (\Sigma, f_1)$, $m_2 = (\Sigma, f_2)$ be multisets over Σ, then the union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset (Σ, g), where $g(x) = f_1(x) + f_2(x)$ for each $x \in \Sigma$. The relative complement of m_2 in m_1, denoted by $m_1 \setminus m_2$, is the multiset (Σ, g), where $g(x) = f_1(x) - f_2(x)$ if $f_1(x) \geq f_2(x)$, and $g(x) = 0$ otherwise.

Definition 1. A P system with proteins on membranes and membrane division of degree $m \geq 1$ is a tuple $\Pi = (O, P, \mu, w_1/z_1, \dots, w_m/z_m, \mathcal{E}, R_1, \dots, R_m, i_{out})$, where:

- O and P are finite non-empty alphabets such that $O \cap P = \emptyset$.
- μ is a singly-rooted tree with m nodes labeled by $1, \dots, m$.
- $w_i, 1 \leq i \leq m$, are multisets over O.
- $z_i, 1 \leq i \leq m$, are multisets over P.
- $\mathcal{E} \subseteq O$ is a finite alphabet.
- $i_{out} \in \{0, 1, \dots, m\}$ is the output region.
- $R_i, 1 \leq i \leq m$, are finite sets of rules of the following types:
  - *Evolution rules* of the types:
    (1) $[\ p \mid a\ ]_i \to [\ p' \mid b\ ]_i$, $p, p' \in P$, $a, b \in O$, $1 \leq i \leq m$;
    (The rule is applicable when protein p is bounded on membrane i and object a appears in membrane i. The application of the rule means that object a evolves to b under the control of the protein p, and at the same time the protein p is modified to p'. Other proteins on membrane i and other objects in membrane i keep unchanged.)
    (2) $a[\ p \mid\ ]_i \to b[\ p' \mid\ ]_i$, $p, p' \in P$, $a, b \in O$, $1 \leq i \leq m$;
    (The rule is applicable when protein p is bounded on membrane i and object a appears in the father of membrane i. The application of the rule means that object a evolves to b under the control of the protein p, and at the same time the protein p is modified to p'.)
    (3) $[\ p \mid a\ ]_i \to b[\ p' \mid\ ]_i$, $p, p' \in P$, $a, b \in O$, $1 \leq i \leq m$;
    (The rule is applicable when protein p is bounded on membrane i and object a appears in membrane i. The application of the rule means that object a is sent out of membrane i evolving to b under the control of the protein p, and at the same time the protein p is modified to p'.)
    (4) $a[\ p \mid\ ]_i \to [\ p' \mid b\ ]_i$, $p, p' \in P$, $a, b \in O$, $1 \leq i \leq m$;
    (The rule is applicable when protein p is bounded on membrane i and object a appears in the father of membrane i. The application of the rule means that object a is sent into membrane i evolving to b under the control of the protein p, and at the same time the protein p is modified to p'.)
    (5) $a[\ p \mid b\ ]_i \to c[\ p' \mid d\ ]_i$, $p, p' \in P$, $a, b, c, d \in O$, $1 \leq i \leq m$.
    (The rule is applicable when protein p is bounded on membrane i and object a appears in the father of membrane i and object b appears in membrane i. The application of the rule means that object a is sent into membrane i evolving to d, object b is sent out of membrane i evolving to c, and at the same time the protein p is modified to p'.)
  - *Division rules:*
    (6) $[\ p \mid\ ]_i \to [\ p' \mid\ ]_i [\ p'' \mid\ ]_i$, $p, p', p'' \in P$, $1 \leq i \leq m$, $i \neq i_{out}$, and i cannot be the root of the tree μ.
    (The rule is applicable when protein p is bounded on membrane i. The application of the rule means that membrane i is divided into two copies of membrane with the same label and the same contents (except for p' and p'') duplicated from the original membrane: objects, proteins and other possible embedded membranes. Note that membrane i can be non-elementary, that is, membrane i may contain other membranes inside.)

The rules of P systems with proteins on membranes and membrane division are used in a non-deterministic maximally parallel way. There has only one restriction: when a membrane is divided, other membrane cannot be applied at that step. The new membranes resulting from division could participate in the interaction with other membranes or the environment by means of evolution rules at the next step if they are not divided once again. The label of a membrane identifies the rules which can be applied to it precisely.

A *configuration* of a P system with proteins on membranes and membrane division is described by the current membrane structure, together with all multisets of objects contained in all membranes, and all multisets of proteins presented on all membranes. The system passes from one configuration to another by a maximally parallel application of rules as described above. Each passage from a configuration to a next configuration is called a *transition*. A configuration is a *halting* one if no rule of the system is applicable to it. A *computation* is a (finite or infinite) sequence of transitions starting in the initial configuration. Only a halting computation gives a result, encoded by the multiset of objects present in the output region.

2.2. Recognizer P systems with proteins on membranes and membrane division

In this subsection, in order to investigate the computational efficiency of membrane systems, the notions from classical computational complexity theory are adapted for membrane computing. Recognizer P systems with proteins on membranes and membrane division are introduced for solving decision problems, which is slightly different from the usual recognizer P systems proposed in [31].

Definition 2. A recognizer P system with proteins on membranes and membrane division of degree $m \geq 1$ is a tuple $\Pi = (O, P, \Sigma, \mu, w_1/z_1, \dots, w_m/z_m, \mathcal{E}, R_1, \dots, R_m, i_{out}, i_{in})$ such that:

- the tuple $(O, P, \mu, w_1/z_1, \dots, w_m/z_m, \mathcal{E}, R_1, \dots, R_m, i_{out})$, is a P system with proteins on membranes and membrane division of degree m such that the output zone is the environment ($i_{out} = 0$), and in this case, i_{out} is usually omitted from the tuple;
- Σ is an (input) alphabet strictly contained in O such that $\mathcal{E} \cap \Sigma = \emptyset$;
- the initial multisets $w_1, \dots, w_m$ are over $O \setminus \Sigma$;
- $i_{in} \in \{1, \dots, m\}$ is the label of a distinguished (input) membrane;
- the working alphabet contains two distinguished elements `yes` and `no`;
- for each multiset w over the input alphabet Σ, the computation of the system Π with input w starts from the configuration of the form $(w_1/z_1, \dots, (w_{i_{in}} + w)/z_{i_{in}}, \dots, w_m/z_m, \mu)$, when the system halts, then either object `yes` or object `no` (but not both) must appear in the environment.

2.3. Recognizer timed P systems with proteins on membranes and membrane division

In this subsection, we start by giving the definition of timed P systems with proteins on membranes and membrane division, and then the notion of recognizer timed P systems with proteins on membranes and membrane division is introduced.

Definition 3. A timed P system with proteins on membranes and membrane division of degree $m \geq 1$ is a pair (Π, e), where Π is a P system with proteins on membranes and membrane division of degree m, and e is a time-mapping of Π, that is, e is a mapping from the finite set of rules $R_1 \cup \cdots \cup R_m$ into the set of natural numbers $\mathbb{N}$, where the numbers in $\mathbb{N}$ represent the execution time for the rules. We denote by $\Pi(e)$ the timed P system with proteins on membranes and membrane division.

A timed P system with proteins on membranes and membrane division $\Pi(e)$ works in the following way: an external clock is assumed, which marks time-units of equal length, starting from instant 0. According to this clock, the step t of computation is defined by the period of time between instant $t-1$ and instant t. If a membrane i contains a rule r (evolution rule or division rule) selected to be executed, then the execution of such rule takes $e(r)$ time units to complete. Therefore, if the execution of a rule r is started at instant j, then such rule is completed at instant $j+e(r)$ and the resulting objects, proteins and membranes (generated by the division rule) become available only at the beginning of step $j+e(r)+1$. If an evolution rule is started, then the occurrences of objects and proteins subject to this rule cannot be subject to other rules until the implementation of the rule completes; if the division rule is started, then the occurrences of proteins and membranes subject to this rule cannot be subject to other rules until the implementation of the rule completes.

Definition 4. A recognizer timed P system with proteins on membranes and membrane division of degree $m \geq 1$ is a tuple (Π, e) where Π is a recognizer P system with proteins on membranes and membrane division of degree m and e is a time-mapping of Π.

2.4. Time-free uniform solutions to decision problems by P systems with proteins on membranes and membrane division

P systems with active membranes have been used to solve the SAT problem in a time-free manner, where rule starting steps (RS-steps, for short) are considered as the computation steps [34], and the number of RS-steps is used to characterize how "fast" a timed P system with active membranes solves a decision problem. In this work, the notion of rule starting steps is also used in timed P systems with proteins on membranes and membrane division.

Definition 5. In timed P systems with proteins on membranes and membrane division, a computation step is called an RS-step if at this step at least one rule starts its execution, that is, steps in which some objects and proteins "start" to evolve or membranes "start" to divide.

A *decision problem*, X, is a pair (I_X, Θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*) and Θ_X is a total Boolean function (that is, predicate) over I_X.

Definition 6. [35] A decision problem $X = (I_X, \Theta_X)$ is solvable in polynomial RS-steps by a family $\mathbf{\Pi} = \{\Pi_n \mid n \in \mathbb{N}\}$ of recognizer P systems with proteins on membranes and membrane division in a time-free manner, if the following holds:

- the family $\mathbf{\Pi}$ is polynomially uniform by Turing machines;
- 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 $\Pi_{s(u)}$;
  - the family $\mathbf{\Pi}$ is time-free sound, time-free complete and time-free polynomially bounded with respect to (X, cod, s).

We also say that the family $\mathbf{\Pi}$ provides an efficient *time-free uniform solution* to the decision problem X.

3. An efficient time-free solution to QSAT problem using P systems with proteins on membranes and membrane division

The QSAT problem (satisfiability of quantified propositional formulas) is a well-known **PSPACE**-complete problem [19], it is defined as follows: *Given the fully quantified formula φ^* associated with a Boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, determine whether or not φ^* is satisfiable.*

A formula as above is of the form

$$\varphi^* = Q_1x_1Q_2x_2 \ldots Q_nx_n(C_1 \wedge C_2 \wedge \cdots \wedge C_m),$$

where each Q_j, $1 \le j \le n$, is either $\forall$ or $\exists$, and each C_i, $1 \le i \le m$, is a *clause* of the form of a disjunction

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

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

$$\varphi^* = Q_1x_1Q_2x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)].$$

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

Theorem 3.1. QSAT *problem can be solved in polynomial RS-steps by a family of P systems with proteins on membranes and membrane division with rules of types (a), (c), (d), (e), (f) in a time-free uniform manner, that is, a P system constructed can solve a family of instances with the same size and correctness of the solution does not depend on the execution time of the involved rules.*

Proof. Let a propositional formula $\varphi^* = Q_1x_1Q_2x_2 \ldots Q_nx_n(C_1 \wedge C_2 \wedge \cdots \wedge C_m)$, $C_i = y_{i,1} \vee \cdots \vee y_{i,p_i}$, for some $m \ge 1$, $p_i \ge 1$, where $y_{i,j} \in \{x_k, \neg x_k \mid 1 \le k \le n\}$, for each $1 \le i \le m, 1 \le j \le p_i$; $Q_k \in \{\forall, \exists\}$, for each $1 \le k \le n$; $\neg x_k$ is the negation of a propositional variable x_k, the two connections $\vee, \wedge$ are *or, and*, respectively.

Let us consider the polynomial time computable function $\langle m, n\rangle = ((m+n)(m+n+1)/2)+n$. It is primitive recursive and bijective from $\mathbb{N}^2$ onto $\mathbb{N}$. In what follows, we construct a family $\mathbf{\Pi} = \{\Pi_t \mid t \in \mathbb{N}\}$, and give the appropriate input multisets, then each system Π_t will solve all instances of QSAT with n variables and m clauses, where $t = \langle m, n\rangle$.

We use the following notations to encode the propositional formula φ^*: $cod(\varphi^*) = \alpha_{1,1} \ldots \alpha_{1,n}\alpha_{2,1} \ldots \alpha_{2,n} \ldots \alpha_{m,1} \ldots \alpha_{m,n}$, where for $1 \le i \le m, 1 \le j \le n$ we have:

$$\alpha_{i,j} = \begin{cases} b_{i,j} & \text{if } x_j \text{ appears in } C_i; \\ b'_{i,j} & \text{if } \neg x_j \text{ appears in } C_i; \\ b''_{i,j} & \text{if } x_j \text{ and } \neg x_j \text{ do not appear in } C_i. \end{cases}$$

For each $m, n \in \mathbb{N}$, we construct the recognizer P system with proteins on membranes and membrane division

$$\Pi_{\langle m,n\rangle} = (O, P, \Sigma, \mu, w_1/z_1, \ldots, w_{2n+3}/z_{2n+3}, \emptyset, R_1, \ldots, R_{2n+3}, i_{in}),$$

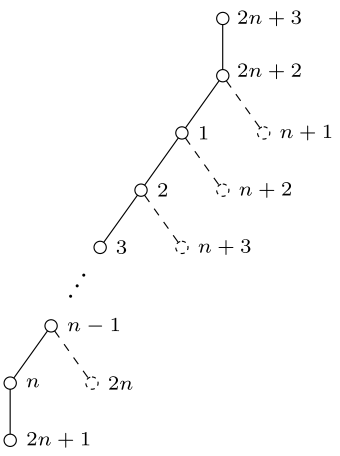

Fig. 1. The initial membrane structure of the P system.

with the following components:

$$
\begin{aligned}
O &= \Sigma \cup \{a_j, a_j^{(1)}, a_j^{(2)}, a_j^{(3)}, t_j, f_j \mid 1 \le i \le n\} \\
&\quad \cup \{d_i \mid 1 \le i \le m\} \cup \{a_{n+1}, a_{n+2}, c, t, t_0, u, \mathtt{yes}, \mathtt{no}\}, \\
P &= \{p_j^+, p_j^-, q_j, q_j', q_j'', r_{m+1,j} \mid 1 \le j \le n\} \cup \{p_k \mid 0 \le k \le n+m+1\} \\
&\quad \cup \{r_{i,j} \mid 1 \le i \le m, 1 \le j \le n\} \cup \{\tilde{p}_0, \bar{p}_0, p_t, r_{1,n+1}, s, s', s''\}, \\
\mu &= [\,[\,[\,[\,\ldots[\,[\,[\;]_{2n+1}\,]_n[\;]_{2n}\,]_{n-1}[\;]_{2n-1}\cdots]_2[\;]_{n+2}\,]_1[\;]_{n+1}\,]_{2n+2}\,]_{2n+3} \\
&\quad \text{(each membrane with label } i \ (1 \le i \le n-1) \text{ contains a membrane} \\
&\quad \text{with label } i+1 \text{ and a membrane with label } n+i+1, \text{ see Fig. 1)}, \\
w_1 &= a_1, w_{2n+3} = \mathtt{no}, w_i = \lambda, 2 \le i \le 2n+2, \\
z_1 &= p_1, z_{2n+1} = r_{1,1}, z_{2n+2} = p_0, z_{2n+3} = s, z_i = p_0, 2 \le i \le n, \\
z_{n+i} &= q_i, 1 \le i \le n, \\
i_{in} &= 2n+1 \text{ (objects } b_{i,j}, b_{i,j}', b_{i,j}'' \text{ are added into this membrane)},
\end{aligned}
$$

and the rules contained in the set $R = \bigcup_{1 \le i \le 2n+3} R_i$ are defined as follows. We divide the computation process into the following four phases: generation phase ($G_{1,j}$–$G_{22,j}$), checking phase (C_1–C_3), quantifier phase ($Q_{1,1}$–$Q_{2,j}$), output phase (O_1–O_4). We also give explanations about the role of these rules in the computation of solving the QSAT problem.

Let e be an arbitrary time-mapping from R to $\mathbb{N}$, where the numbers represent the execution time of the rules from R.

Generation phase.

In generation phase, the system assigns truth-assignment of each variable x_j ($1 \le j \le n$) as well as looking for the clauses satisfied by the truth-assignment of this variable x_j. Specifically, the computation process of assigning truth-assignment of variable x_j as well as looking for the clauses satisfied of this variable is described as follows. The protein p_j ($1 \le j \le n$) on membrane j corresponds to variable x_i. With the appearance of protein p_j on membrane j, a non-elementary membrane with label j is divided into two membranes with the same label, the truth values *true* (represented by t_j) and *false* (represented by f_j) assigned to variable x_j are produced in two separate copies of membrane j. The objects t_j and f_j are sent to membrane n, the clauses which are satisfiable by the corresponding truth-assignment are checked in membranes with label $2n+1$. After that, objects $a_j^{(1)}$ and $a_j^{(2)}$ are produced and sent to membrane $j+1$, and the protein p_{j+1} is produced on membrane $j+1$. Hence the system starts to assign truth-assignment of variable x_{j+1} as well as looking for the clauses satisfied by the truth-assignment of variable x_{j+1}.

In the initial configuration of the system, there is object a_1 in membrane 1 and protein p_1 on membrane 1; protein p_0 on membrane $2n+2$ and on membranes i ($2 \le i \le n$), protein q_i on membrane $n+i$ ($1 \le i \le n$); input multiset $cod(\varphi)$ in membrane $2n+1$ and protein $r_{1,1}$ on membrane $2n+1$, object no in membrane $2n+3$ and protein s on membrane $2n+3$.

Rules $G_{1,j}$–$G_{5,j,k}$ are used to generate 2^n copies of membrane with label n, object t_i (resp., f_i) is sent into membrane n.

$$G_{1,j} : [\,p_j \mid\,]_j \rightarrow [\,p_j^+ \mid\,]_j[\,p_j^- \mid\,]_j, \ 1 \le j \le n.$$

$$G_{2,j} : [\,p_j^+ \mid a_j\,]_j \rightarrow [\,p_j^+ \mid t_j\,]_j, \ 1 \le j \le n.$$

$G_{3,j} : [\, p_j^- \mid a_j\,]_j \to [\, p_j^- \mid f_j\,]_j,\ 1 \le j \le n.$

$G_{4,j,k} : t_j[\, p_0 \mid\,]_k \to [\, p_0 \mid t_j\,]_k,\ 1 \le j \le n-1, j+1 \le k \le n.$

$G_{5,j,k} : f_j[\, p_0 \mid\,]_k \to [\, p_0 \mid f_j\,]_k,\ 1 \le j \le n-1, j+1 \le k \le n.$

At step 1, division rule $G_{1,1}$ is applied, under the influence of protein p_1, membrane 1 is divided into two copies of membrane 1, with protein p_1 replaced by p_1^+ and p_1^-, respectively. For any given time-mapping e, the execution of rule $G_{1,1}$ completes in $e(G_{1,1})$ steps. Exception for the application of rule $G_{1,1}$, at step 1, rule O_2 is also used, where object no is sent out of membrane $2n+3$, and changing the protein from s to s'. From step 2 to step $e(G_{1,1})$, there is no rule starting. So, during the execution of rule $G_{1,1}$ (i.e., from step 1 to step $e(G_{1,1})$), there is one RS-step. Note that the number of RS-steps during the execution of rule $G_{1,1}$ is independent on the time-mapping e.

After the execution of rule $G_{1,1}$ finishes, under the control of proteins p_1^+ and p_1^- placed on two separate copies of membrane 1, the applications of rules $G_{2,1}$ and $G_{3,1}$ start at the same step, but they may complete at different steps due to the fact that the execution time associated with rules $G_{2,1}$ and $G_{3,1}$ can be different. By applying rule $G_{2,1}$ (resp., $G_{3,1}$), object a_1 is evolved to t_1 (resp., f_1).

With the object t_1 (resp., f_1) in membrane 1 and the protein p_0 on membrane 2, rule $G_{4,1,2}$ (resp., $G_{5,1,2}$) is enabled and used, object t_1 (resp., f_1) is sent into membrane 2. When the execution of rule $G_{4,1,2}$ (resp., $G_{5,1,2}$) finishes, under the control of protein p_0 on membrane 3, the application of rule $G_{4,1,3}$ (resp., $G_{5,1,3}$) starts, object t_1 (resp., f_1) is sent into membrane 3. In this way, under the influence of protein p_0 on membranes i $(2 \le i \le n)$, rules $G_{4,1,k}$ (resp., $G_{5,1,k}$) $(2 \le k \le n)$ are applied one by one, where object t_1 is transferred into the membrane n. Note that the executions of rules $G_{4,1,2}$ and $G_{5,1,2}$ may start at different steps since rules $G_{2,1}$ and $G_{3,1}$ may complete at different steps; rules $G_{4,1,k}$ and $G_{5,1,k}$ $(3 \le k \le n)$ may start at different steps because the executions of rules $G_{4,1,k-1}$ and $G_{5,1,k-1}$ may finish at different steps.

Rules $G_{6,i,j}$–$G_{9,i,j}$ are used to check the clauses which are satisfiable by the corresponding truth-assignment.

$$\begin{aligned}
G_{6,i,j} : \{ & t_j[\, r_{i,j} \mid b_{i,j}\,]_{2n+1} \to d_i[\, r_{i+1,j} \mid t_j\,]_{2n+1},\\
& t_j[\, r_{i,j} \mid b'_{i,j}\,]_{2n+1} \to c[\, r_{i+1,j} \mid t_j\,]_{2n+1},\\
& t_j[\, r_{i,j} \mid b''_{i,j}\,]_{2n+1} \to c[\, r_{i+1,j} \mid t_j\,]_{2n+1} \mid 1 \le i \le m, 1 \le j \le n\}.\\
G_{7,i,j} : \{ & f_j[\, r_{i,j} \mid b_{i,j}\,]_{2n+1} \to c[\, r_{i+1,j} \mid f_j\,]_{2n+1},\\
& f_j[\, r_{i,j} \mid b'_{i,j}\,]_{2n+1} \to d_i[\, r_{i+1,j} \mid f_j\,]_{2n+1},\\
& f_j[\, r_{i,j} \mid b''_{i,j}\,]_{2n+1} \to c[\, r_{i+1,j} \mid f_j\,]_{2n+1} \mid 1 \le i \le m, 1 \le j \le n\}.
\end{aligned}$$

$G_{8,i,j} : [\, r_{i,j} \mid t_j\,]_{2n+1} \to t_j[\, r_{i,j} \mid\,]_{2n+1},\ 2 \le i \le m, 1 \le j \le n.$

$G_{9,i,j} : [\, r_{i,j} \mid f_j\,]_{2n+1} \to f_j[\, r_{i,j} \mid\,]_{2n+1},\ 2 \le i \le m, 1 \le j \le n.$

When the execution of rule $G_{4,1,n}$ (resp., $G_{5,1,n}$) completes, and with protein $r_{1,1}$ on membrane $2n+1$, one of rules in $G_{6,1,1}$ (resp., $G_{7,1,1}$) is enabled, which corresponds to checking whether the first clause is satisfied by the truth-assignment *true* (resp., *false*) of variable x_1. By using one of rules in $G_{6,1,1}$ (resp., $G_{7,1,1}$), object d_1 can be generated in membrane n if object $b_{1,1}$ (resp., $b'_{1,1}$) appears in membrane $2n+1$, otherwise, object c presents in membrane n. Meanwhile, protein $r_{1,1}$ placed on membrane $2n+1$ is changed to $r_{2,1}$. If object t_1 (resp., f_1) appears in membrane $2n+1$, and with protein $r_{2,1}$ on membrane $2n+1$, the application of rule $G_{8,2,1}$ (resp., $G_{9,2,1}$) starts, object t_1 (resp., f_1) is sent out of membrane $2n+1$. After the execution of rule $G_{8,2,1}$ (resp., $G_{9,2,1}$) completes, under the control of protein $r_{2,1}$ on membrane $2n+1$, one of rules in $G_{6,2,1}$ (resp., $G_{7,2,1}$) is enabled, which corresponds to check whether the second clause is satisfied by the truth-assignment *true* (resp., *false*) of variable x_1. By applying one of rules in $G_{6,2,1}$ (resp., $G_{7,2,1}$), object d_2 can be generated in membrane n if object $b_{2,1}$ (resp., $b'_{2,1}$) presents in membrane $2n+1$, otherwise, object c appears in membrane n; protein $r_{2,1}$ placed on membrane $2n+1$ is changed to $r_{3,1}$. With object t_1 (resp., f_1) in membrane $2n+1$ and protein $r_{3,1}$ on this membrane, the application of rule $G_{8,3,1}$ (resp., $G_{9,3,1}$) starts, object t_1 (resp., f_1) is sent out of membrane $2n+1$. When the execution of rule $G_{8,3,1}$ (resp., $G_{9,3,1}$) finishes, and with protein $r_{3,1}$ on membrane $2n+1$, one of rules in $G_{6,3,1}$ (resp., $G_{7,3,1}$) is enabled, the system starts to check whether the third clause is satisfied by the truth-assignment *true* (resp., *false*) of variable x_1. In this way, rules $G_{6,i,1}$ $(1 \le i \le m)$ and $G_{8,i+1,1}$ $(1 \le i \le m-1)$ (resp., $G_{7,i,1}$ $(1 \le i \le m)$ and $G_{9,i+1,1}$ $(1 \le i \le m-1)$) are applied one by one until the protein $r_{m+1,1}$ presents on membranes $2n+1$. Rules $G_{6,1,1}$ and $G_{7,1,1}$ may start at different steps since the executions of rules $G_{4,1,n}$ and $G_{5,1,n}$ may finish at different steps; rules $G_{6,i,1}$ and $G_{7,i,1}$ $(2 \le i \le m)$ may start at different steps since the executions of rules $G_{8,i,1}$ and $G_{9,i,1}$ may finish at different steps.

Rules $G_{10,j}$–$G_{13,j,k}$ are used to send object $a_j^{(1)}$ (resp., $a_j^{(2)}$) to membrane j.

$G_{10,j} : [\, r_{m+1,j} \mid t_j\,]_{2n+1} \to a_j^{(1)}[\, r_{1,j+1} \mid\,]_{2n+1},\ 1 \le j \le n.$

$G_{11,j} : [\, r_{m+1,j} \mid f_j\,]_{2n+1} \to a_j^{(2)}[\, r_{1,j+1} \mid\,]_{2n+1},\ 1 \le j \le n.$

$$G_{12,j,k} : [\, p_0 \mid a_j^{(1)} \,]_k \to a_j^{(1)}[\, p_0 \mid \,]_k,\ 1 \le j \le n-1,\ j+1 \le k \le n.$$

$$G_{13,j,k} : [\, p_0 \mid a_j^{(2)} \,]_k \to a_j^{(2)}[\, p_0 \mid \,]_k,\ 1 \le j \le n-1,\ j+1 \le k \le n.$$

With the appearance of protein $r_{m+1,1}$ on membrane $2n+1$, rule $G_{10,1}$ (resp., $G_{11,1}$) is enabled and used, object t_1 (resp., f_1) is evolved to $a_1^{(1)}$ (resp., $a_1^{(2)}$), and $a_1^{(1)}$ (resp., $a_1^{(2)}$) is sent out of membrane $2n+1$, changing the protein from $r_{m+1,1}$ to $r_{1,2}$. If object $a_1^{(1)}$ (resp., $a_1^{(2)}$) presents in membrane n, under the control of protein p_0 on membrane n, the application of rule $G_{12,1,n}$ (resp., $G_{13,1,n}$) starts, object $a_1^{(1)}$ (resp., $a_1^{(2)}$) is sent out of membrane n. With object $a_1^{(1)}$ (resp., $a_1^{(2)}$) in membrane $n-1$ and protein p_0 on membrane $n-1$, rule $G_{12,1,n-1}$ (resp., $G_{13,1,n-1}$) is enabled and applied, object $a_1^{(1)}$ (resp., $a_1^{(2)}$) is sent out of membrane $n-1$. In this way, rules $G_{12,1,k}$ (resp., $G_{13,1,k}$) $(2 \le k \le n)$ are used one by one, object $a_1^{(1)}$ (resp., $a_1^{(2)}$) is transferred to the membrane 1. Rules $G_{10,1}$ and $G_{11,1}$ may start at different steps since the executions of rules $G_{6,m,1}$ and $G_{7,m,1}$ may finish at different steps; rules $G_{12,1,n}$ and $G_{13,1,n}$ may start at different steps since the executions of rules $G_{10,1}$ and $G_{11,1}$ may finish at different steps; the executions of rules $G_{12,1,k}$ and $G_{13,1,k}$ $(2 \le k \le n-1)$ may start at different steps since rules $G_{12,1,k+1}$ and $G_{13,1,k+1}$ may finish at different steps.

Rules $G_{14,j}$–$G_{16,j}$ are used to make the system synchronization.

$$G_{14,j} : [\, p_j^+ \mid a_j^{(1)} \,]_j \to a_j^{(1)}[\, p_j^- \mid \,]_j,\ 1 \le j \le n.$$

$$G_{15,j} : [\, p_j^- \mid a_j^{(2)} \,]_j \to a_j^{(2)}[\, p_j^+ \mid \,]_j,\ 1 \le j \le n.$$

$$G_{16,j} : a_j^{(2)}[\, p_j^- \mid \,]_j \to [\, p_j^- \mid a_j^{(3)} \,]_j,\ 1 \le j \le n.$$

When the execution of rule $G_{12,1,2}$ (resp., $G_{13,1,2}$) finishes, with protein p_1^+ (resp., p_1^-) on membrane 1, the application of rule $G_{14,1}$ (resp., $G_{15,1}$) starts, object $a_1^{(1)}$ (resp., $a_1^{(2)}$) is sent out of membrane 1, changing protein from p_1^+ (resp., p_1^-) to p_1^- (resp., p_1^+). The executions of rules $G_{14,1}$ and $G_{15,1}$ may start at different steps due to the fact that rules $G_{12,1,2}$ and $G_{13,1,2}$ may finish at different steps. Rule $G_{16,1}$ is enabled only when object $a_1^{(2)}$ appears in membrane $2n+2$ and there is a membrane 1 having protein p_1^- on membrane 1, that is, rule $G_{16,1}$ can be used only when the applications of both rules $G_{14,1}$ and $G_{15,1}$ have completed. By using rule $G_{16,1}$, object $a_1^{(2)}$ evolves to $a_1^{(3)}$, and object $a_1^{(3)}$ is sent into membrane 1. Note that rule $G_{16,1}$ has a synchronization function because $e(G_{2,1}) + \Sigma_{2 \le k \le n} e(G_{4,1,k}) + \Sigma_{1 \le i \le m} e(G_{6,i,1}) + \Sigma_{2 \le i \le m} e(G_{8,i,1}) + e(G_{10,1}) + \Sigma_{2 \le k \le n} e(G_{12,1,k}) + e(G_{14,1})$ may not equal to $e(G_{3,1}) + \Sigma_{2 \le k \le n} e(G_{5,1,k}) + \Sigma_{1 \le i \le m} e(G_{7,i,1}) + \Sigma_{2 \le i \le m} e(G_{9,i,1}) + e(G_{11,1}) + \Sigma_{2 \le k \le n} e(G_{13,1,k}) + e(G_{15,1})$. In any case, when the execution of rule $G_{16,1}$ completes, the computation takes at most $4n+4m+1$ RS-steps, which is independent on any time-mapping e.

Rules $G_{17,j}$–$G_{22,j}$ are used to generate object a_{j+1} in membrane $j+1$.

$$G_{17,j} : [\, p_j^- \mid a_j^{(3)} \,]_j \to a_j^{(3)}[\, p_j^+ \mid \,]_j,\ 1 \le j \le n.$$

$$G_{18,j} : a_j^{(3)}[\, q_j \mid \,]_{n+j} \to [\, q_j' \mid a_{j+1} \,]_{n+j},\ 1 \le j \le n.$$

$$G_{19,j} : [\, q_j' \mid \,]_{n+j} \to [\, q_j'' \mid \,]_{n+j}[\, q_j'' \mid \,]_{n+j},\ 1 \le j \le n.$$

$$G_{20,j} : [\, q_j'' \mid a_{j+1} \,]_{n+j} \to a_{j+1}[\, q_j'' \mid \,]_{n+j},\ 1 \le j \le n.$$

$$G_{21,j} : a_{j+1}[\, p_j^+ \mid \,]_j \to [\, p_0 \mid a_{j+1} \,]_j,\ 1 \le j \le n.$$

$$G_{22,j} : a_j[\, p_0 \mid \,]_j \to [\, p_j \mid a_j \,]_j,\ 2 \le j \le n.$$

With object $a_1^{(3)}$ in membrane 1 and protein p_1^- on membrane 1, rule $G_{17,1}$ is enabled and applied, object $a_1^{(3)}$ is sent out of membrane 1, changing protein from p_1^- to p_1^+. When the execution of rule $G_{17,1}$ finishes and with protein q_1 on membrane $n+1$, the application of rule $G_{18,1}$ starts, object $a_1^{(3)}$ is evolved to a_2, and object a_2 is sent into membrane $n+1$, changing protein from q_1 to q_1'. With the appearance of protein q_1' on membrane $n+1$, rule $G_{19,1}$ is enabled and applied, membrane $n+1$ is divided into two copies of membrane $n+1$, protein q_1' is replaced by q_1'' on each copy of membrane $n+1$. The function of rule $G_{19,1}$ is used to double the number of object a_2. If protein q_1'' presents on membrane $n+1$, the application of rule $G_{20,1}$ in all membranes $n+1$ starts at the same time and completes at the same time due to the fact that the execution of rule $G_{20,1}$ in all membranes $n+1$ takes the same time $e(G_{20,1})$ for a given time-mapping e. By using rule $G_{20,1}$, object a_2 is sent out of membrane $n+1$. When the execution of rule $G_{20,1}$ finishes, with object a_2 in membrane $2n+2$ and protein p_1^+ on membrane 1, the application of rule $G_{21,1}$ starts, each object a_2 is sent into a membrane with label 1, changing protein from p_1^+ to p_0 (there are two copies of object a_2 in membrane $2n+2$ and two membranes 1 with protein p_1^+ placed on, and the system works in a maximally parallel manner). With object a_2 in membrane 1 and protein p_0 on membrane 2, rule $G_{22,2}$ is enabled and applied, object a_2 is sent into membrane 2, changing protein from p_0 to p_2. Note that the application of rule $G_{22,2}$ in all membranes 2 starts and completes at the same time due to the fact that the execution of rule $G_{22,2}$ in all membranes 2 takes the same time $e(G_{22,2})$ for a given time-mapping e.

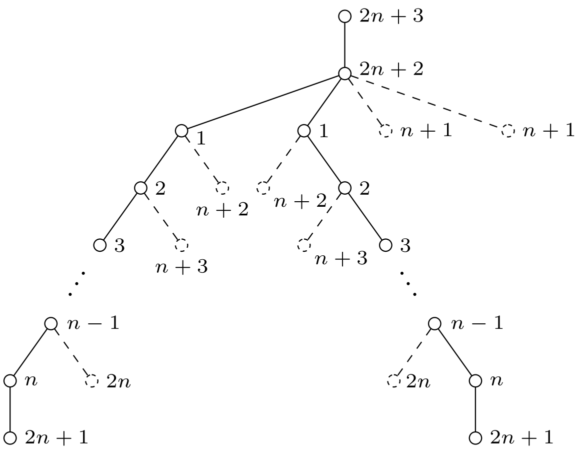

Fig. 2. The membrane structure at the moment when protein p_2 appears on membrane 2.

In general, when protein p_2 presents on membrane 2 (the object a_2 appears in membrane 2), the computation takes at most $4n+4m+7$ RS-steps (see Fig. 2).

With protein p_2 presents on membrane 2, rule $G_{1,2}$ is enabled, which means that the system starts to assign truth-assignment of variable x_2, and look for the clauses satisfied by the truth-assignment of variable x_2. Note that the application of rule $G_{1,2}$ in all membranes 2 starts at the same time and completes at the same time since the execution of rule $G_{1,2}$ in all membranes 2 takes the same time $e(G_{1,2})$ for a given time-mapping e. The executions of rules $G_{2,2}$ and $G_{3,2}$ start at the same step, but they may complete at different steps. Furthermore, the applications of rules $G_{4,2,k}, G_{5,2,k}, G_{12,2,k}, G_{13,2,k}$ $(3 \le k \le n)$, $G_{6,i,2}$ $(1 \le i \le m)$, $G_{7,i,2}$ $(1 \le i \le m)$, $G_{8,i,2}$ $(2 \le i \le m)$, $G_{9,i,2}$ $(2 \le i \le m)$, $G_{10,2}, G_{11,2}, G_{14,2}, G_{15,2}$ may start and complete at different steps. However, rule $G_{16,2}$ is enabled only when all rules that have started and completed their executions. Similar with the case of variable x_1, the processes of assigning truth-assignment of variable x_2 as well as looking for the clauses satisfied by the truth-assignment of variable x_2 take at most $4n+4m+3$ RS-steps. Comparing with the processes of assigning truth-assignment of variable x_1 as well as looking for the clauses satisfied by the truth-assignment of variable x_1, where the computation takes at most $4n+4m+7$ RS-steps, this is because during the computation, objects t_1 and f_1 enter the membranes from membrane 1 to membrane n, and objects $a_1^{(1)}$ and $a_1^{(2)}$ are sent out of membranes from membrane n to membrane 1; however, the processes of assigning truth-assignment of variable x_2 as well as looking for the clauses satisfied by the truth-assignment of variable x_2, objects t_2 and f_2 are sent into membranes from membrane 2 to membrane n, and objects $a_2^{(1)}$ and $a_2^{(2)}$ are sent out of membranes from membrane n to membrane 2. Analogously, we can deduce that if we consider the processes of assigning truth-assignment as well as looking for the clauses satisfied by the truth-assignment of variables $x_1, x_2, \dots, x_n$ in the worse case, then with the increasing of the system assigns truth-assignment as well as looks for the clauses satisfied by the truth-assignment of variable from x_i to x_{i+1} $(1 \le i \le n-1)$, the number of RS-steps decreases by 4. Hence, it is easy to see that the processes of assigning truth-assignment of variable x_n as well as looking for the clauses satisfied by the truth-assignment of variable x_n take at most $4m+11$ RS-steps.

Therefore, after at most $2n^2+4nm+9n$ RS-steps, 2^n separate copies of membrane with label n are generated, each membrane with label n contains a membrane with label $2n+1$. Moreover, membrane $2n+2$ contains two copies of membrane with label $n+1$ and each membrane with label j $(1 \le j \le n-1)$ contains two copies of membrane with label $n+j+1$ (these membranes with labels $n+j$ $(1 \le j \le n)$ are idle in the subsequent phases, thus, we do not show them in Fig. 3). All the membranes are placed in the membrane with label $2n+3$ (see Fig. 3 for the main membrane structure at the moment when the generation phase completes).

Checking phase.

When the generation phase completes, each membrane with label n contains some of the objects from the set $\{d_1, d_2, \dots, d_m\}$ whose elements denote the corresponding clauses satisfied by the truth assignment of the variables. If there is at least one membrane with label n that contains all the objects $d_1, d_2, \dots, d_m$, which means the formula φ without quantifiers is satisfied by the corresponding truth assignment in that membrane; if there are no membranes with label n that contain all the objects $d_1, d_2, \dots, d_m$, the formula φ without quantifiers is not satisfied.

$$C_1 : [\ p_0 \mid a_{n+1}\ ]_n \rightarrow [\ p_{n+1} \mid a_{n+2}\ ]_n.$$

$$C_{2,i} : [\ p_{n+i} \mid d_i\ ]_n \rightarrow d_i[\ p_{n+i+1} \mid\ ]_n,\ \ 1 \le i \le m.$$

$$C_3 : [\ p_{n+m+1} \mid a_{n+2}\ ]_n \rightarrow [\ \tilde{p}_0 \mid t\ ]_n.$$

When the execution of rule $G_{20,n}$ completes, with protein p_n^+ on membrane n, all objects a_{n+1} in each membrane $n-1$ are sent into membrane n, changing protein from p_n^+ to p_0 at the same step (there are two copies of object a_{n+1} and two

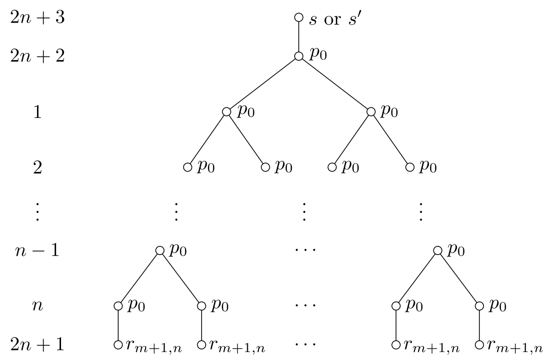

Fig. 3. The main membrane structure of the system when the generation phase completes. The symbols at nodes indicate the proteins placed on membranes, depth levels $2n+3, 2n+2, 1, \ldots, n, 2n+1$ correspond to labels of membranes.

copies of membrane with label n in each membrane $n-1$; each object a_{n+1} is sent into a membrane n by using rule $G_{21,n}$ in a maximally parallel manner). So, object a_{n+1} appears in each membrane with label n at the same step. By applying rule C_1, object a_{n+1} is evolved to a_{n+2}, and protein p_0 placed on membrane n is changed to p_{n+1}. After the execution of rule C_1 completes, the system starts to check whether object d_1 appears in each membrane with label n.

With protein p_{n+1} on membrane n, if object d_1 appears in membrane n, the application of rule $C_{2,1}$ starts, object d_1 is sent out of membrane n, and changing protein from p_{n+1} to p_{n+2}. When the execution of rule $C_{2,1}$ finishes, the system starts to check whether object d_2 appears in a membrane n (i.e., checking whether the corresponding true assignment satisfies the clause C_2). The proteins p_{n+3} appear only when the corresponding membranes n contain object d_2.

The system continues to check whether objects $d_3, d_4, \ldots, d_m$ appear in membranes n. For any membrane n that does not contain object d_i, $i = 1, 2, \ldots, m$, then the computation in this membrane stops at the time when $C_{2,i}$ is supposed to be applied. If protein p_{n+m+1} appears on membrane with label n, rule C_3 is enabled and used, object a_{n+2} is evolved to t, protein p_{n+m+1} placed on membrane n is changed to $\tilde{p}_0$.

In general, the checking phase takes at most $m+2$ RS-steps.

Quantifier phase.

A membrane with label j corresponds to the quantifier Q_j, where $1 \le i \le n$. If $Q_j = \forall$, $2 \le j \le n$ (resp., $j = 1$) a single object t will appear in membrane $j-1$ (resp., $2n+2$) only when two copies of object t_0 (each lower level membrane provides one copy of object t_0) appear in membrane $j-1$ (resp., $2n+2$), that is, the respective clauses are satisfied for both truth values of x_j. If $Q_j = \exists$, $2 \le j \le n$ (resp., $j = 1$) a single object t_0 appeared in membrane $j-1$ (resp., $2n+2$) is enough, and one copy of object t will appear in membrane $j-1$ (resp., $2n+2$).

Note that the rules designed for quantifier phase are suitable for any cases of quantifier Q_j ($1 \le i \le n$). However, for a specific QSAT problem, each variable x_j corresponds to a specific quantifier Q_j, that is, for checking whether variable x_j with the quantifier Q_j is satisfied, only one multiset of rules ($(o_{1,j,1}, o_{1,j,2}, o_{1,j,3})$ or $(o_{2,j,1}, o_{2,j,2})$) is assigned to the corresponding membranes.

$$\begin{aligned}
&\text{If } Q_1 = \forall, \text{ we have rules:}\\
&\qquad o_{1,1,1} : [\, \tilde{p}_0 \mid t \,]_1 \to t_0[\, p_t \mid \,]_1,\\
&\qquad o_{1,1,2} : [\, p_0 \mid t_0 \,]_{2n+2} \to [\, \bar{p}_0 \mid u \,]_{2n+2},\\
&\qquad o_{1,1,3} : [\, \bar{p}_0 \mid t_0 \,]_{2n+2} \to [\, \tilde{p}_0 \mid t \,]_{2n+2}.\\
&\text{If } Q_j = \forall\ (2 \le j \le n), \text{ we have rules:}\\
&\qquad o_{1,j,1} : [\, \tilde{p}_0 \mid t \,]_j \to t_0[\, p_t \mid \,]_j,\\
&\qquad o_{1,j,2} : [\, p_0 \mid t_0 \,]_{j-1} \to [\, \bar{p}_0 \mid u \,]_{j-1},\\
&\qquad o_{1,j,3} : [\, \bar{p}_0 \mid t_0 \,]_{j-1} \to [\, \tilde{p}_0 \mid t \,]_{j-1}.\\
&\text{If } Q_1 = \exists, \text{ we have rules:}\\
&\qquad o_{2,1,1} : [\, \tilde{p}_0 \mid t \,]_1 \to t_0[\, p_t \mid \,]_1,\\
&\qquad o_{2,1,2} : [\, p_0 \mid t_0 \,]_{2n+2} \to [\, \tilde{p}_0 \mid t \,]_{2n+2}.\\
&\text{If } Q_j = \exists\ (2 \le j \le n), \text{ we have rules:}\\
&\qquad o_{2,j,1} : [\, \tilde{p}_0 \mid t \,]_j \to t_0[\, p_t \mid \,]_j,\\
&\qquad o_{2,j,2} : [\, p_0 \mid t_0 \,]_{j-1} \to [\, \tilde{p}_0 \mid t \,]_{j-1}.
\end{aligned}$$

The appearance of object t in membrane n means that the checking phase finishes and the quantifier phase starts. In what follows, we describe how the system simulates quantifiers $\forall$ and $\exists$.

If $Q_j = \forall$ $(2 \le j \le n)$, then we simulate the quantifier $\forall$ by using the rules $o_{1,j,1}, o_{1,j,2}, o_{1,j,3}$. Specifically, when the object t appears in membrane j and the protein $\tilde{p}_0$ on membrane j, rule $o_{1,j,1}$ is enabled and used, object t is sent out of membrane j and evolved to object t_0, changing the protein from $\tilde{p}_0$ to p_t. Note that the application of rule $o_{1,j,1}$ in membranes j starts and completes at the same step due to the fact that the execution of such rule in membranes j takes the same time for a given time-mapping e. When object t_0 presents in a membrane $j-1$ and protein p_0 appears on this membrane, the application of rule $o_{1,j,2}$ starts, object t_0 is evolved to the "dummy" object u, changing the protein from p_0 to $\bar{p}_0$. With the protein $\bar{p}_0$ presents on membrane $j-1$, rule $o_{1,j,3}$ is enabled and used, object t_0 is evolved to t, changing the protein from $\bar{p}_0$ to $\tilde{p}_0$. Note that the object t presents in membranes $j-1$ at the same time since the executions of rules $o_{1,j,2}$ and $o_{1,j,3}$ in membranes $j-1$ take the same time for a given time-mapping e. If $Q_1 = \forall$, it can be simulated by using the rules $o_{1,1,1}, o_{1,1,2}, o_{1,1,3}$, one copy of object t will appear in membrane $2n+2$ only when two copies of object t_0 appear in membrane $2n+2$. The simulation process is similar to the case of $Q_j = \forall$ $(2 \le j \le n)$.

If $Q_j = \exists$ $(2 \le j \le n)$, then we simulate the quantifier $\exists$ by using rules $o_{2,j,1}, o_{2,j,2}$. Specifically, with object t in membrane j and protein $\tilde{p}_0$ on membrane j, rule $o_{2,j,1}$ is enabled and used, object t is evolved to t_0, and object t_0 is sent out of membrane $j-1$, changing protein from $\tilde{p}_0$ to p_t. Note that the application of rule $o_{2,j,1}$ in membranes j starts and completes at the same step due to the fact that the execution of such rule in membranes j takes the same time for a given time-mapping e. With object t_0 in membrane $j-1$ and protein p_0 on membrane $j-1$, the application of rule $o_{2,j,2}$ starts, object t_0 is evolved to t, changing protein from p_0 to $\tilde{p}_0$. The execution of rule $o_{2,j,2}$ in membranes $j-1$ starts and completes at the same step since the execution of such rule in membranes $j-1$ takes the same time for a given time-mapping e. If $Q_1 = \exists$, it can be simulated by using rules $o_{2,1,1}, o_{2,1,2}$, one copy of object t_0 presented in membrane with label $2n+2$ is enough. The simulation process is similar to the case of $Q_j = \exists$ $(2 \le j \le n)$.

Supposed that the number of quantifier $\forall$ of formula φ^* is k, and the simulation of each quantifier $\forall$ (resp., quantifier $\exists$) of the formula takes three (resp., two) RS-steps, thus, the simulation of all quantifiers $\forall$ (resp., quantifier $\exists$) of the formula takes $3k$ RS-steps (resp., $2(n-k)$ RS-steps).

Output phase.

The system sends to the environment the right answer. If formula φ^* is satisfiable, object $\mathtt{yes}$ appears in the environment when the system halts; if formula φ^* is not satisfiable, object $\mathtt{no}$ appears in the environment when the system halts.

$$O_1 : [\, \tilde{p}_0 \mid t \,]_{2n+2} \rightarrow t[\, p_t \mid \,]_{2n+2}.$$

$$O_2 : [\, s \mid \mathtt{no} \,]_{2n+3} \rightarrow \mathtt{no}[\, s' \mid \,]_{2n+3}.$$

$$O_3 : [\, s' \mid t \,]_{2n+3} \rightarrow \mathtt{yes}[\, s'' \mid \,]_{2n+3}.$$

$$O_4 : \mathtt{no}[\, s'' \mid \,]_{2n+3} \rightarrow [\, s'' \mid \mathtt{no} \,]_{2n+3}.$$

In the initial configuration of the system, membrane $2n+3$ contains object $\mathtt{no}$. At step 1, rule O_2 is applied, where object $\mathtt{no}$ is sent out of the membrane $2n+3$, and changing protein from s to s'. Note that the application of rule O_2 takes no RS-step. When the quantifier phase finishes, we have the following two cases.

- If no object t presents in membrane with label $2n+2$, then rules O_1, O_3, O_4 can not be applied. In this case, when the system halts, object $\mathtt{no}$ remains in the environment, which means that the formula is not satisfiable.
- If object t appears in membrane $2n+2$, under the influence of protein $\tilde{p}_0$, rule O_1 is enabled and used, object t is sent out of membrane $2n+2$, and changing its protein from $\tilde{p}_0$ to p_t. When the execution of rule O_1 finishes, at this moment, if the execution of rule O_2 is not yet finished, then no rule can be started in the system. Only when the execution of rule O_2 completes, protein on membrane $2n+3$ is changed to s', rule O_3 will be enabled. By applying rule O_3, object t is evolved to $\mathtt{yes}$, and object $\mathtt{yes}$ is sent out of membrane $2n+3$, changing protein from s' to s''. With the appearance of protein s'' on membrane $2n+3$, the application of rule O_4 starts, object $\mathtt{no}$ is sent into membrane $2n+3$. In this case, when the system halts, object $\mathtt{yes}$ remains in the environment, which means that the formula is satisfiable. This process takes three RS-steps.

According to the explanation of the P system as above mentioned, it is easy to see that for any time-mapping $e : R \rightarrow \mathbb{N}$, when the computation halts, object $\mathtt{yes}$ presents in the environment if and only if the formula φ^* is satisfiable; and when the computation halts, object $\mathtt{no}$ presents in the environment if and only if the formula φ^* is not satisfiable. Hence the system $\Pi_{\langle m,n \rangle}$ is time-free sound and time-free complete.

For any time-mapping $e : R \rightarrow \mathbb{N}$, if the formula φ^* is satisfiable, the computation takes at most $2n^2 + 4nm + 11n + m + k + 5$ RS-steps: it takes at most $2n^2 + 4nm + 9n$ RS-steps to generate 2^n membranes with label n (2^n different truth-assignments); it takes at most $m+2$ RS-steps to check whether all clauses without quantifiers are satisfied by a truth-assignment; the quantifier phase takes $2n+k$ RS-steps; and the output phase takes three RS-steps, the system halts. If the formula φ^* is not satisfiable, the computation takes at most $2n^2 + 4nm + 11n + m + k + 2$ RS-steps (it takes no RS-step at the output phrase), and the system halts. Therefore, the family of P systems with proteins on membranes is time-free polynomially bounded.

The family $\mathbf{\Pi} = \{\Pi_{\langle m,n\rangle} \mid m, n \in \mathbb{N}\}$ defined above is polynomially uniform by Turing machines because the construction of P system is built in polynomial time with respect to the size parameters n and m, and necessary resources are as follows:

- size of the object set: $3nm + 6n + m + 8$;
- size of the protein set: $nm + 7n + m + 9$;
- initial number of membranes: $2n + 3$;
- total size of initial multisets of objects: $nm + 2$;
- initial number of proteins on membranes: $2n + 3$;
- number of rules: $2n^2 + 8nm + 15n + m + 5$;
- the maximal length of a rule including proteins: 6.

Therefore, the QSAT problem can be solved in polynomial RS-steps by a family of recognizer P systems with proteins on membranes and membrane division in a time-free uniform manner. □

4. Conclusions and further works

In this work, inspired by the fact that biochemical reactions in biological systems are inherently parallel and have different reaction rates, and the execution time of biochemical reactions is sensitive to environmental factors that could affect reaction rates in an unpredictable way, we investigate the computational efficiency of timed P systems with proteins on membranes. Specifically, we construct a family of P systems proteins on membranes and membrane division, which solves QSAT problem in a time-free uniform manner, where a P system constructed can solve a family of instances with the same size and correctness of the solution does not depend on the execution time of the involved rules.

The P system constructed in the proof of Theorem 3.1 has the rules of types (a), (c), (d), (e), (f). It is interesting to design P systems with fewer types of rules for efficiently solving the QSAT problem. Moreover, division rules for non-elementary membranes are used in Theorem 3.1, it remains open whether the QSAT problem can be solved by P systems with proteins on membranes and using division rules for elementary membranes in a time-free manner.

Flip-flop membrane systems with proteins are a particular class of P systems with proteins on membranes [23]. In such P systems, each protein has at most two states: p and $\bar{p}$, and the rules are used either without changing the protein or changing the protein from p to $\bar{p}$ and back. It remains open whether we can give a time-free uniform solution to the QSAT problem by using flip-flop membrane systems with proteins.

Small universal P systems have been widely studied [10,18,21]. In P systems with proteins on membranes, small universal P systems working in a maximally parallel manner have also been studied in subsection 5.5 [32]. It is of interest to construct small universal P systems with proteins on membranes in the context of time-freeness.

Recently, a strategy of using rules, called flat maximal parallelism was considered in [16,36], where in each step, a maximal set of applicable rules is chosen and each rule in the set is applied exactly once in each membrane. It is of interest to investigate the computational power of P systems with protein on membranes by using rules in a minimally parallel way or in a flat maximally parallel way in the context of time-freeness.

The biological phenomenon of membrane fission process (also called membrane separation) was incorporated in membrane computing, which has been investigated in the framework of P system [14,17]. It is interesting to investigate the computational power of P systems with protein on cells and cell separation in a time-free manner.

Acknowledgments

The work was supported by National Natural Science Foundation of China (61602192, 61320106005 and 61033003), Ph.D. Programs Foundation of Ministry of Education of China (20120142130008), China Postdoctoral Science Foundation (2016M600592, 2017T100554), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, 4th ed., Garland Science, New York, 2002.
[2] A. Alhazov, R. Freund, Variants of small universal P systems with catalysts, Fundam. Inform. 138 (1–2) (2015) 227–250.
[3] R. Brijder, M. Cavaliere, A. Riscos-Núñez, G. Rozenberg, D. Sburlan, Membrane systems with marked membranes, Electron. Notes Theor. Comput. Sci. 171 (2007) 25–36.
[4] L. Cardelli, Brane calculi-interactions of biological membranes, in: Proceedings of the International Conference of Computational Methods in Systems Biology, CMSB 2004, in: Lect. Notes Comput. Sci., vol. 3082, Springer, 2005, pp. 257–278.
[5] L. Cardelli, Gh. Păun, An universality result for a (mem)brane calculus based on mate/drip operations, Int. J. Found. Comput. Sci. 17 (2006) 49–68.
[6] M. Cavaliere, D. Sburlan, Time-independent P systems, in: Proceedings of the 5th International Workshop, WMC 2004, in: Lect. Notes Comput. Sci., vol. 3365, Springer, 2005, pp. 239–258.
[7] M. Cavaliere, S. Sedwards, Modelling cellular processes using membrane systems with peripheral and integral proteins, in: Proceedings of International Conference of Computational Methods in Systems Biology, CMSB 2006, in: Lect. Notes Comput. Sci., vol. 4210, Springer, 2006, pp. 108–126.

[8] M. Cavaliere, S. Sedwards, Membrane systems with peripheral proteins: transport and evolution, Electron. Notes Theor. Comput. Sci. 171 (2) (2007) 37–53.
[9] M. Cavaliere, S. Sedwards, Decision problems in membrane systems with peripheral proteins, transport and evolution, Theor. Comput. Sci. 404 (1–2) (2008) 40–51.
[10] E. Csuhaj-Varjú, M. Margenstern, G. Vaszil, S. Verlan, On small universal antiport P systems, Theor. Comput. Sci. 372 (2) (2007) 152–164.
[11] M. Cavaliere, Time-free solutions to hard computational problems, in: Research Frontiers of Membrane Computing: Open Problems and Research Topics, M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez (Eds.), Int. J. Found. Comput. Sci. 24 (5) (2013) 579–582, Section 11.
[12] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2–3) (2006) 279–308.
[13] S. Krishna, On the computational power of flip-flop proteins on membranes, in: Proceedings of the 3rd Conference on Computability in Europe, in: Lect. Notes Comput. Sci., vol. 4497, Springer, 2007, pp. 695–704.
[14] 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.
[15] C. Martin-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Patón, Tissue P systems, Theor. Comput. Sci. 296 (2) (2003) 295–326.
[16] L. Pan, Gh. Păun, B. Song, Flat maximal parallelism in P systems with promoters, Theor. Comput. Sci. 623 (2016) 83–91.
[17] L. Pan, M.J. Pérez-Jiménez, Computational complexity of tissue-like P systems, J. Complex. 26 (3) (2010) 296–315.
[18] L. Pan, X. Zeng, Small universal spiking neural P systems working in exhaustive mode, IEEE Trans. Nanobiosci. 10 (2) (2011) 99–105.
[19] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, Reading, MA, 1994.
[20] A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, New Gener. Comput. 20 (3) (2002) 295–306.
[21] A. Păun, Gh. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 48–60.
[22] A. Păun, M. Păun, A. Rodríguez-Patón, M. Sidoroff, P systems with proteins on membranes: a survey, Int. J. Found. Comput. Sci. 22 (1) (2011) 39–53.
[23] A. Păun, B. Popa, P systems with proteins on membranes, Fundam. Inform. 72 (2006) 467–483.
[24] A. Păun, B. Popa, P systems with proteins on membranes and membrane division, in: Proceedings of the 10th International Conference, DLT 2006, in: Lect. Notes Comput. Sci., vol. 4036, Springer, 2006, pp. 292–303.
[25] A. Păun, A. Rodríuez-Patón, On flip-flop membrane systems with proteins, in: Proceedings of the 8th International Workshop, WMC 2007, in: Lect. Notes Comput. Sci., vol. 4860, Springer, 2007, pp. 414–427.
[26] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[27] Gh. Păun, One more universality result for P systems with objects on membranes, Int. J. Comput. Commun. Control 1 (1) (2006) 44–51.
[28] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, New York, 2010.
[29] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, Inf. Sci. 304 (2015) 80–91.
[30] 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.
[31] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, A polynomial complexity class in P systems using membrane division, J. Autom. Lang. Comb. 11 (4) (2006) 423–434.
[32] B. Popa, Membranes Systems with Limited Parallelism, PhD Thesis, Louisiana Tech. Univ., Ruston, USA, 2006.
[33] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 3, Springer-Verlag, Berlin, 1997.
[34] T. Song, L.F. Macías-Ramos, L. Pan, M.J. Pérez-Jiménez, Time-free solution to SAT problem using P systems with active membranes, Theor. Comput. Sci. 529 (2014) 61–68.
[35] B. Song, M.J. Pérez-Jiménez, L. Pan, An efficient time-free solution to SAT problem by P systems with proteins on membranes, J. Comput. Syst. Sci. 82 (2016) 1090–1099.
[36] 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.
[37] T. Song, L. Pan, Spiking neural P systems with request rules, Neurocomputing 193 (2016) 193–200.
[38] 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.
[39] B. Song, C. Zhang, L. Pan, Tissue-like P systems with evolutional symport/antiport rules, Inf. Sci. 378 (2017) 177–193.
[40] P. Sosík, M. Langer, Small (purely) catalytic P systems simulating register machines, Theor. Comput. Sci. 623 (2016) 65–74.
[41] P. Sosík, A. Păun, A. Rodríguez-Patón, P systems with proteins on membranes characterize PSPACE, Theor. Comput. Sci. 488 (2013) 78–95.
[42] T. Wu, Z. Zhang, Gh. Păun, L. Pan, Cell-like spiking neural P systems, Theor. Comput. Sci. 623 (2016) 180–189.
[43] 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.
[44] G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, Int. J. Neural Syst. 24 (5) (2014) 1–16.