Math. Struct. in Comp. Science (2019), *vol.* 29, *pp.* 663–680. © Cambridge University Press 2018
doi:10.1017/S0960129518000282 First published online 9 August 2018

The computational power of timed P systems with active membranes using promoters

YUEGUO LUO[†], HAIJUN TAN[†], YING ZHANG[†] and YUN JIANG[‡]

[†]*College of Computer Engineering, Yangtze Normal University, Chongqing 408100, China*
Email: ygluo@cqu.edu.cn, 21249892@qq.com, 77101258@qq.com
[‡]*School of Artificial Intelligence, Chongqing Technology and Business University, Chongqing 400067, China*
Email: 34976532@qq.com

Received 1 August 2017; revised 3 March 2018

P systems with active membranes are a class of bioinspired computing models, where the rules are used in the non-deterministic maximally parallel manner. In this paper, first, a new variant of timed P systems with active membranes is proposed, where the application of rules can be regulated by promoters with only two polarizations. Next, we prove that any Turing computable set of numbers can be generated by such a P system in the time-free way. Moreover, we construct a uniform solution to the $\mathcal{SAT}$ problem in the framework of such recognizer timed P systems in polynomial time, and the feasibility and effectiveness of the proposed system is demonstrated by an instance. Compared with the existing methods, the P systems constructed in our work require fewer necessary resources and RS-steps, which show that the solution is effective to **NP**-complete problem.

1. Introduction

Membrane computing is a new area of computer science in recent years. As we know, life is directly related to cells, and organism consists of cells. A cell is fragile and robust in a way to control the biochemical processes during billions of years of evolution (Păun 1999a). As a pioneering work, Păun conceived the model of membrane computing (Păun 2000). Therefore, membrane systems are usually called P systems. Until now, there are mainly three classes of P systems that have been introduced: cell-like P systems, tissue P systems and neural-like P systems (Ionescu et al. 2006). Many different variants of P systems have been proved to be Turing universal when used as number computing devices (Pan et al. 2011; Song et al. 2016a; Song and Pan 2016), language generators (Wu et al. 2016; Zeng et al. 2014a; Zhang et al. 2014) and function computing devices (Păun et al. 2007; Zeng et al. 2014b; Zhang et al. 2017). In recent years, application areas in the framework of P systems are also studied (Zhang et al. 2012, 2014). Inspired by some biological facts, several methods have been proposed to produce new membranes in living cells: membrane division (Păun et al. 2004, 2008), membrane creation (Gutiérrez-Naranjo et al. 2007) and membrane separation (Pan et al. 2010; Pérez-Jiménez et al. 2015).

Cell-like P systems are a class of computing models motivated by biochemical interactions between membranes in a cell-like arrangement (Păun 2000). Each membrane

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

contains a class of objects and some rules. These rules are mainly evolution rules and division rules. Membranes can communicate with each other by using evolution rules. With mathematical and biological motivations, some variants of cell-like P systems have been proposed (Alhazov et al. 2004; Păun et al. 2002). As an important variant, P systems with active membranes were presented in Păun (1999b). In such a P system, the polarization of each membrane can be positive (represented by +), negative (represented by −) or neutral (represented by 0). By applying rules, the polarization of corresponding membrane can be changed. In addition, an important feature of P systems with active membranes is that the membrane structure may change when applying division rules or dissolution rules.

In this paper, we will pay close attention to P systems with active membranes. Based on this model, the $\mathcal{SAT}$ problem has been solved effectively (see Guo et al. 2015; Luo et al. 2015, 2016; Leporati et al. 2009). But in these papers, it is assumed that each rule can be completed in exactly one unit time in a standard P system with active membranes. However, the execution time of different biochemical reactions is often influenced by a variety of factors so that its execution time is hard to control. Therefore, it is necessary to construct P systems that work independently from the values associated with the execution time of the rules. The concept of time-free was introduced in Cavaliere et al. (2005), and some open problems were proposed in Gheorghe et al. (2013). Recently, some **NP**-hard problems have been solved in a time-free manner (see Song et al. 2014, 2015a, 2015b, 2015c, 2016b, 2017a, 2017b). It needs to be emphasized that the $\mathcal{SAT}$ problem has been solved by timed P systems with active membranes (Song et al. 2014, 2015a, 2015b). In Song et al. (2014, 2015b), the P systems are constructed in a semi-uniform way, that is, P systems are constructed from the instances. Compared with the P systems constructed in Song et al. (2014), new membranes have the same labels with their parent membranes in Song et al. (2015b). In Song et al. (2015a), however, the P systems are constructed in a uniform way.

As a variant of the original model, the model of P systems with promoters was proposed in Bottoni et al. (2002). Based on this method, in this work, we proposed a new variant of timed P systems with active membranes, called P systems with active membranes using promoters (*AMPP* systems, for short), which is inspired by the biological fact that some biochemical reactions may be controlled by promoters in the P systems with active membranes. Thus, rules are considered as the biochemistry of living cells, where reactions take place under the influence of some chemical compounds. The chemical compounds can be regarded as promoters. In this way, the application of rules in the membranes is regulated by promoters.

In this paper, we investigate the computational power of *AMPP* systems in the time-free way. In addition, we simplify the systems of active membranes, and only consider the systems with two membrane polarizations (namely, + and −). Specifically, take advantage of membrane division, we solve the $\mathcal{SAT}$ problem by active membranes with promoters and two membrane polarizations in the time-free way. Furthermore, the P systems are constructed in a uniform way, and in a time-free manner. Compared with semi-uniform way, P systems with uniform way can solve all instances of a given size.

The paper is organized as follows: first, we propose a new variant of timed P systems with active membranes. Then, we prove that any Turing computable set of numbers can

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

be generated by an *AMPP* system in the time-free way. Next, in Section 4, we give a time-free uniform solution to the $\mathcal{SAT}$ problem with *AMPP* systems. The feasibility and effectiveness of the proposed system is demonstrated by an instance in Section 5. Finally, some formal details and conclusions are presented, and some open problems are proposed in the end.

2. *AMPP* systems

Definition 1. Formally, an *AMPP* system of degree $m \geqslant 1$ is a tuple of the form:

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R, i_{out}),$$

where

— O is the finite non-empty alphabet of objects;
— H is a finite set of membrane labels;
— μ is an initial membrane structure with membranes labeled with elements in H;
— $w_i, 1 \leqslant i \leqslant m$, is a string over O^* representing the multiset of objects in membrane with label i;
— i_{out} is output region of membrane structure, and it saves the results;
— R is a finite set of developmental rules with the following types ($\mathcal{PRO}$ denote a promoter in a membrane):

a. $[a \to u|_{\mathcal{PRO}}]_h^{e_0}, h \in H,\ e_0 \in \{+,-\},\ a, \mathcal{PRO} \in O, u \in O^*$.
Object evolution rules; the application of this rule starts to a configuration in the presence of promoter $\mathcal{PRO}$ and object a. When the application of this rule finishes, multiset u can be generated and promoter $\mathcal{PRO}$ cannot be changed. If the promoter of this rule is empty, the rule can be written to $[a \to u]_h^{e_0}$ instead of $[a \to u|_{\mathcal{PRO}}]_h^{e_0}$;

b. $a[]_h^{e_1} \to [b]_h^{e_2}, h \in H,\ e_1, e_2 \in \{+,-\},\ a, b \in O$.
Communication rules; an object a enters the membrane and may be modified. At the same time, the polarization of the membrane can be changed, but its label keeps unchanged;

c. $[a]_h^{e_1} \to []_h^{e_2} b, h \in H,\ e_1, e_2 \in \{+,-\},\ a, b \in O$.
Communication rules; an object a is sent out of the membrane and may be modified. At the same time, the polarization of the membrane can be changed, but its label keeps unchanged;

d. $[a]_h^{e_0} \to b, h \in H,\ e_0 \in \{+,-\},\ a, b \in O$.
Dissolution rules; under the influence of the object present in a membrane, the membrane can be dissolved. At the same time, the object in that membrane can be modified;

e. $[a]_h^{e_1} \to [b]_h^{e_2}[c]_h^{e_3}, h \in H,\ e_1, e_2, e_3 \in \{+,-\},\ a, b, c \in O$.

Division rules for elementary membranes; under the influence of the object present in a membrane, this rule can be applied to divided the membrane into two copies of membranes (with the same label h); object b and object c in the two membranes can be

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

generated, respectively. The remaining objects in membrane h enter the newly generated membranes, respectively.

The following rules are the extended rules of type (e):

$(e^{(1)})$ $[a]_{h_1}^{e_1} \to [b]_{h_2}^{e_2}[c]_{h_3}^{e_3}, h_1, h_2, h_3 \in H,\ e_1, e_2, e_3 \in \{+, -\},\ a, b, c \in O.$

Division rules for elementary membranes; by applying this rule, the newly generated membranes can have different labels, which is different from the rule type (e).

$(e^{(2)})$ $[a]_h^{e_1} \to [b]_h^{e_2}[c]_h^{e_3}, h \in H,\ e_1, e_2, e_3 \in \{+, -\},\ a, b, c \in O.$

Division rules for non-elementary membranes; in this case, there may be one membrane or many membranes in a membrane with label h. By applying this rule, the newly generated membranes have the same label with initial membrane (with the same label h). All remaining objects and all membranes in the membrane h remain unchanged, and enter the newly generated membranes, respectively.

In the case of using promoters, a rule can be applied when a certain object appears in a certain membrane. But promoters do not participate in the application of rules, and a promoter may enable application of several rules at the same time. For example, there is a rule $u \to v|_p$, where object p is a promoter. This rule can be executed if object p appears in the located region. In this case, however, a promoter cannot be changed during the execution of the rule. On the contrary, this rule cannot be executed if object p is absent in the located region. Hence, the function of promoter is that it starts the application of a rule. Furthermore, a promoter can guide more than one rule to be executed at the same time. In particular, promoters themselves can evolve to other objects according to corresponding rules.

The initial configuration is represented by $(w_1, \ldots, w_m, \mu)$, that is, objects denoted by $(w_1, \ldots, w_m)$ are placed in corresponding membranes. Then, when the execution of one or more rules finishes, we can get transitions between the configurations. At each step, the current configuration is described by multisets of objects and membrane structure. If there exists a halting computation, finally, the P systems will stop running. At that moment, there is no rule that is being executed, and no rule can be applied. In each membrane, it must be stressed that rules are applied according to the principle of maximal parallelism, that is, at each step, if no any other rules can be added to apply, all rules that can be applied have to be applied to all possible objects.

Definition 2. A timed *AMPP* system is a pair (Π, e), where Π is an *AMPP* system, and e denote the execution time of the rules. We denote the timed *AMPP* system with Π(e).

Definition 3. A recognizer timed *AMPP* system (of degree $m \geqslant 1$) can be defined as the form:

$$\Pi = (O, H, \Sigma, \mu, w_1, \ldots, w_m, R, e, i_{in}, i_{out}),$$

where

— $(O, H, \mu, w_1, \ldots, w_m, R, i_{out})$ is a P system;
— Σ is an (input) alphabet;
— μ is an initial membrane structure with membranes labeled with elements of H;

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

— $w_i, 1 \leqslant i \leqslant m$, is a finite multiset over O^*;
— R is a finite set of rules;
— We can specify the execution time of each rule by a mapping $e : R \to \mathbb{N}$, where $\mathbb{N}$ is the set of non-negative integers;
— $i_{in} \in \{1, 2, ..., m\}$ is the input membrane;
— i_{out} is the environment, which saves the results;
— The working alphabet contains two distinguished elements $\mathcal{YES}$ and $\mathcal{NO}$;
— All the computations halt;
— When the system halts, either object $\mathcal{YES}$ or $\mathcal{NO}$ must appear in the environment.

The input multiset has been added to the contents of the input membrane i_{in}. For the multiset w over the input alphabet Σ, the computation of the system with input w starts as the form:

$$(w_1, \ldots (w_{i_{in}} + w), \ldots w_m, \mu).$$

Finally, a recognizer timed *AMPP* system has to halt, and the object in the environment is computing result in the stopping configuration. If the object $\mathcal{YES}$ appears in the environment when the system stops, it is an *accepting computation*; on the contrary, if the object $\mathcal{NO}$ appears, it is a *rejecting computation.*

Definition 4. Let $X = (I_X, \theta_X)$ be a decision problem (with $\mathcal{YES}/\mathcal{NO}$ answer), where I_X is a set of instances and θ_X is a predicate over I_X , and let $\mathbf{\Pi} = \{\Pi(n) | n \in \mathbb{N}\}$ be a family of recognizer P systems. We use *rule starting step* (RS-step, for short) to define the computation step in timed P systems (for more details, please see Song et al. 2014).

Definition 5. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time and in uniform way by a family $\mathbf{\Pi} = \{\Pi(n) | n \in \mathbb{N}\}$ of recognizer timed *AMPP* systems, if the following holds:

— The family Π is polynomially uniform with respect to Turing machines; namely, there exists a deterministic Turing machine that constructs the system $\Pi(n)$ with knowledge involving only the size of the problem X for every instance of X working in polynomial time;
— 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 $\Pi(s(n))$;
  – The family Π is time-free sound with regard to (X, *cod*, *s*). For each instance $u \in I_X$ such that there exists an accepting computation of if $\Pi(s(u), e)$ with input $cod(u)$, we have $\theta_x(u) = 1$;
  – The family Π is time-free complete with regard to (X, *cod*, *s*). For each instance $u \in I_X$, if $\theta_x(u) = 1$, then every computation of $\Pi(s(u), e)$ with input $cod(u)$ is an accepting one for any time-mapping e;
  – The family Π is time-free polynomially bounded with regard to (X, *cod*, *s*); namly, there exists a polynomial function $p(n)$ such that for each $u \in I_X$, when all the computations in $\Pi(s(u), e)$ with input $cod(u)$ halts, there is at most $p(|u|)$ RS-steps for any time-mapping e.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

3. Universality of time-free *AMPP* systems

In this section, by simulating register machines, we prove that *AMPP* systems are universal.

Definition 6. A register machine is a tuple $M = (m, H, l_0, l_h, I)$, where

— m is the number of registers;
— H is a set of instruction labels;
— $l_0, l_h \in H$, l_0 is the initial label, and l_h is the halting label;
— I is a set of labeled program instructions of the following forms:
  – $l_i : (ADD(r), l_j, l_k)$ add 1 to register r, and then the instruction with labels l_j or l_k is applied non-deterministically;
  – $l_i : (SUB(r), l_j, l_k)$ if register is non-zero, then subtract 1 from register r. Next, the instruction with label l_j can be applied; otherwise, the instruction with label l_k is applied;
  – $l_h : HALT$.

A set $N(M)$ of numbers can be generated by a register machine M. Next, we introduce how it works. Initially, all registers are empty. The computation of the machine starts by applying the instruction l_0. When the halt instruction l_h is applied, then at that moment number n will present in the specified register 1. We can say that the number n is generated by the machine M. Therefore, in this case, the machine M can generate a set of numbers, which can be denoted by $N(M)$. Only when the computation stops, the system can generate numbers. If the computation does not halt, then no number can be generated.

It is known that register machines compute all sets of numbers that are Turing computable; hence, they characterize *NRE* (Minsky 1967).

Theorem 1. $NOP_4^f((a),(b),(c)) = NRE$.

Where NOP_m^f denotes the families of the sets of natural numbers generated by timed *AMPP* systems (m represents the number of initial membranes, and f represents time-free).

Proof. $M = (m, H, l_0, l_h, I)$ is a register machine with three registers. We construct a P system to simulate the register machine in a time-free way. The P system can be defined as follows:

$$\Pi = (O, L, \mu, w_1, \ldots, w_4, R, i_{out}),$$

where

— $O = \{a\} \cup \{l, l', l^{(2)}, l^{(3)}, l^{(4)} | l \in H\}$;
— L =1, 2, 3, 4;
— $\mu = [[\ \ ]_1^+[\ \ ]_2^+[\ \ ]_3^+]_4^+$;
— $w_1 = w_2 = w_3 = \lambda,\ w_4 = l_0$;
— i_{out}=1.

Each membrane is associated with label r (register r=1, 2, 3), and the number of object a in membrane r represents the value of the corresponding register r. We can specify the execution time of each rule by a mapping $e : I \to \mathbb{N}$, where I is the set of rules.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

1. For each *ADD* instruction $l_i : (ADD(r), l_j, l_k)$, $r=1, 2, 3$, the simulation uses the rules:
   $R_1 : l_i[\ \ ]_r^+ \rightarrow [l'_i]_r^+$.
   $R_2 : [l'_i \rightarrow l_i^{(2)} a]_r^+$.
   $R_3 : [l_i^{(2)}]_r^+ \rightarrow l_j[\ \ ]_r^+$.
   $R_4 : [l_i^{(2)}]_r^+ \rightarrow l_k[\ \ ]_r^+$.
   At a step, object l_i enters the correct membrane r and evolves to l_i' by applying rule R_1. When object occurs in membrane r, by applying rule R_2, object a can be generated in membrane r as required by the *ADD* instruction l_i. At the same time, object $l_i^{(2)}$ appears in membrane r. At that moment, rules R_3 and R_4 can be chosen non-deterministically. Thus, object $l_i^{(2)}$ will be sent out of that membrane and object l_j (resp., l_k) can be generated by applying the rule R_3 (resp., R_4). It should be pointed out that the computation result is the same for difference time-mapping e, that is, it is independent of the time-mapping e.
2. For each *SUB* instruction $l_i : (SUB(r), l_j, l_k)$, $r=1, 2, 3$, the simulation with the set I uses the rules:
   $R_5 : l_i[\ \ ]_r^+ \rightarrow [l'_i]_r^+$.
   $R_6 : [l'_i]_r^+ \rightarrow [\ \ ]_r^- l_i^{(2)}$.
   $R_7 : [a]_r^- \rightarrow a[\ \ ]_r^+$.
   $R_8 : [l_i^{(2)}]_4^+ \rightarrow [\ \ ]_4^- l_i^{(3)}$.
   $R_9 : l_i^{(3)}[]_4^- \rightarrow [l_i^{(4)}]_4^+$.
   $R_{10} : [a \rightarrow l_j|_{l_i^{(4)}}]_4^+$.
   $R_{11} : l_i^{(4)}[\ \ ]_r^- \rightarrow [l_k]_r^-$.
   $R_{12} : [l_k]_r^- \rightarrow l_k[\ \ ]_r^+$.

At step 1, the system starts running to simulate the *SUB* instruction l_i, and rule R_5 can be applied. Object l_i enters the correct membrane r, and evolves to l_i'. At the following step, object l_i' can be sent out of that membrane and evolves to $l_i^{(2)}$. At that moment, there will be the following two cases.

1. When the *SUB* instruction l_i starts to simulate, there exists at least one copy of object a in the membrane r. When the execution of the rule R_6 is completed, the rules R_7 and R_8 are applied simultaneously. Since the polarization of membrane r is negative, an object a can be sent out of that membrane by applying the rule R_7, and the charge of that membrane is changed from negative to positive. In this way, an object a will be consumed in the membrane r. In addition, by executing the rule R_8, the object $l_i^{(2)}$ is sent to the membrane 4 and evolves to $l_i^{(3)}$, and the charge of that membrane is changed from positive to negative. At the following step, rule R_9 start to be applied, the object $l_i^{(3)}$ enters the membrane 4 and evolves to $l_i^{(4)}$, and the charge of that membrane is changed to positive. When the object $l_i^{(4)}$ is present in the membrane 4, the object a evolves to l_j with the promoter $l_i^{(4)}$. Thus, the system starts to simulate the instruction l_j.
2. When the *SUB* instruction l_i starts to simulate, there is no object a in the membrane r. In this case, the rules R_8 and R_9 will be applied, and object $l_i^{(4)}$ appears in the membrane 4. In this case, by applying the rule R_{11}, the object $l_i^{(4)}$ enters the membrane r and evolves to object l_k. When the object l_k appears in the membrane r, l_k is sent

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

out of that membrane by applying the rule R_{12}, and the charge of that membrane is changed to positive. Thus, the system starts to simulate the instruction l_j.

4. A time-free uniform solution to $\mathcal{SAT}$ problem with *AMPP* systems

Definition 7. The $\mathcal{SAT}$ problem is as follows: given a Boolean formula in conjunctive normal form, to determine whether or not it is satisfiable, that is, whether there exists an assignment to its variables on which it evaluates to true.

The $\mathcal{SAT}$ problem is a well-known **NP**-complete problem. This problem has been widely used in artificial intelligence and computer theory. In this work, we present a uniform solution to the $\mathcal{SAT}$ problem. The computing process consists of the following phases:

— Generation phase: all assignments of variables can be generated by using membrane division;
— Checking phase: the system checks whether or not the formula have a truth assignment. If all assignments of variables cannot satisfy the clause, it means that there is no satisfiable solution; or it means that there is at least one satisfiable solution;
— Output phase: the object $\mathcal{YES}/\mathcal{NO}$ in the environment is computing result in the stopping configuration.

Theorem 2. The $\mathcal{SAT}$ problem can be solved by a uniform family of timed *AMPP* systems with rules of types (a), (b), (c), $(e^{(2)})$ by membrane division in polynomial time.

Proof. For a formula with n Boolean variables and m clauses. Let us consider a propositional formula $\gamma = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, consisting of m clauses $C_j = y_{1,j} \vee \ldots \vee y_{p_j,j}$, where $y_{i,j} \in \{x_l,\ \neg x_l | 1 \leqslant l \leqslant n\}$, $1 \leqslant i \leqslant p_j$, $1 \leqslant j \leqslant m$; $\neg x_l$ is the negation of a propositional variable x_l.

Let us consider the polynomial-time computable function (the pair function)

$$g(n,k) = ((n+k)(n+k+1)/2) + n.$$

It is a primitive recursive and bijective function from $\mathbb{N}^2$ to $\mathbb{N}$.

The instance γ is encoded by multiset as follows:

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

We codify γ , where for $1 \leqslant j \leqslant m$, $1 \leqslant i \leqslant n$, by the multiset

$$\alpha_{i,j} = \begin{cases} Y_{i,j} & \text{if } x_i \text{ appears in } \mathrm{C}_j; \\ N_{i,j} & \text{if } \neg x_i \text{ appears in } \mathrm{C}_j; \\ B_{i,j} & \text{both } x_i \text{ and } \neg x_i \text{ do not appear in } \mathrm{C}_j. \end{cases}$$

Next, we define a family of recognizer *AMPP* systems, which can process all instances γ provided the input multiset $cod(\gamma)$ to P systems. For the given n and m, a recognizer *AMPP* system is constructed

$$\Pi_{\mathcal{SAT}(m,n)} = (O, \Sigma, H, \mu, w_1, \ldots, w_m, R, e, i_{in}, i_{out}),$$

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

where

— $O = \Sigma \cup \{a_i | 1 \leqslant i \leqslant n+1\} \cup \{t_{i,j}, f_{i,j}, e_i | 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant 2m+1\}$
$\cup \{r_j | 1 \leqslant j \leqslant m\} \cup \{s_j | 2 \leqslant j \leqslant m+1\} \cup \{\mathcal{YES}, \mathcal{NO}\}$;
— $\Sigma = \{Y_{i,j}, N_{i,j}, B_{i,j} | 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$;
— $H = \{1, 2, n+1, n+2, \ldots, n+m\}$ is the set of labels of the membranes;
— $\mu = [[[\ \ ]^-_{n+1}[\ \ ]^-_{n+2} \ldots [\ \ ]^-_{n+m}]^+_2]^+_1$ is initial membrane structure;
— $w_i, 1 \leqslant i \leqslant m+n$, is a string over O^*, describing the multisets of objects placed in membranes:

$$w_1 = \mathcal{NO},\ w_2 = a_1,\ w_i = \lambda,\ n+1 \leqslant i \leqslant m+n;$$

— $i_{in} = 2$ is the input membrane; $i_{out} = 0$ is the environment that saves the results;
— We can specify the execution time of each rule by a mapping $e : R \to \mathbb{N}$, where R is the set of rules. For a rule R_i in R, its execution time can be expressed as $e(R_i)$. R is the following set of rules:

1. Generation phase
$R_{1,i} : [a_i]^+_2 \to [t_{i,1}]^-_2[f_{i,1}]^-_2,\ 1 \leqslant \mathrm{i} \leqslant n.$
$R_{2,i,j} : [Y_{i,j} \to r_j e_i|_{t_{i,1}}]^-_2,\ [N_{i,j} \to e_i|_{t_{i,1}}]^-_2,\ [B_{i,j} \to e_i|_{t_{i,1}}]^-_2,\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{3,i,j} : [Y_{i,j} \to e_i|_{f_{i,1}}]^-_2,\ [N_{i,j} \to r_j e_i|_{f_{i,1}}]^-_2,\ [B_{i,j} \to e_i|_{f_{i,1}}]^-_2,\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{4,i,j} : e_i[\ \ ]^-_{n+j} \to [e_i]^+_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{5,i,j} : t_{i,j}[\ \ ]^+_{n+j} \to [t_{i,j}]^+_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{6,i,j} : f_{i,j}[\ \ ]^+_{n+j} \to [f_{i,j}]^+_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{7,i,j} : [t_{i,j}]^+_{n+j} \to t_{i,j+1}[\ \ ]^+_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{8,i,j} : [f_{i,j}]^+_{n+j} \to f_{i,j+1}[\ \ ]^+_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{9,i,j} : t_{i,m+j}[\ \ ]^+_{n+j} \to [t_{i,m+j}]^-_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{10,i,j} : f_{i,m+j}[\ \ ]^+_{n+j} \to [f_{i,m+j}]^-_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{11,i,j} : [t_{i,m+j}]^-_{n+j} \to t_{i,m+j+1}[\ \ ]^-_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{12,i,j} : [f_{i,m+j}]^-_{n+j} \to f_{i,m+j+1}[\ \ ]^-_{n+j},\ 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m.$
$R_{13,i} : [t_{i,2m+1}]^-_2 \to t_{i,2m+1}[\ \ ]^-_2,\ 1 \leqslant i \leqslant n.$
$R_{14,i} : [f_{i,2m+1}]^-_2 \to f_{i,2m+1}[\ \ ]^-_2,\ 1 \leqslant i \leqslant n.$
$R_{15,i} : [f_{i,2m+1} \to a^2_{i+1}|_{\mathrm{t}_{i,2m+1}}]^+_1,\ 1 \leqslant i \leqslant n.$
$R_{16,i} : a_i[\ \ ]^-_2 \to [a_i]^+_2,\ 2 \leqslant i \leqslant n+1.$
2. Checking phase
$R_{17} : [r_1 \to s_2|_{a_{n+1}}]^+_2.$
$R_{18,j} : [r_j \to \mathrm{s}_{j+1}|_{s_j}]^+_2,\ 2 \leqslant j \leqslant m.$
3. Output phase
$R_{19} : [\mathcal{NO}]^+_1 \to \mathcal{NO}[\ \ ]^+_1.$
$R_{20} : [a_{n+1}|_{\mathrm{s}_{m+1}} \to \mathcal{YES}]^+_2.$
$R_{21} : [\mathcal{YES}]^+_2 \to \mathcal{YES}[\ \ ]^+_2.$
$R_{22} : [\mathcal{YES}]^+_1 \to \mathcal{YES}[\ \ ]^-_1.$
$R_{23} : \mathcal{NO}[\ \ ]^-_1 \to [\mathcal{NO}]^-_1.$

In what follows, we will give an overview of computing process to show how the constructed *AMPP* systems work.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

Generation phase:

Let e be the execution time of the rules. Initially, membrane 1 contains object $\mathcal{NO}$, cod(γ); membrane 2 contains object a_1.

In general, the object a_i in the membrane with label 2 corresponds to variable x_i, $1 \leqslant i \leqslant n$. At step 1, the system starts running, and rules $R_{1,1}$ and R_{19} can be applied at the same time. Rule $R_{1,1}$ is a division rule, which can be applied to divide membrane 2 into two membranes with the same label. Simultaneously, object a_1 evolves to objects $t_{1,1}$ and $f_{1,1}$ in the generated new membranes (object $t_{1,1}$ corresponds to assignment *true* of variable x_1; object $f_{1,1}$ corresponds to assignment *false* of variable x_1). In this process, there is just one RS-step because no other rules can be applied except for the rules $R_{1,1}$ and R_{19}. When the application of rule $R_{1,1}$ finishes, rules $R_{2,1,j}$ and $R_{3,1,j}$ ($1 \leqslant i \leqslant m$) can be enabled at the same time. These rules correspond to check whether or not each clause is satisfied by the assignments of variable x_1. By applying the rule $R_{2,1,j}$ (resp., $R_{3,1,j}$), under the influence of the promoter $t_{1,1}$ (resp., $f_{1,1}$), object $Y_{1,j}$ (resp., $N_{1,j}$) evolves to object r_j and e_1. In other cases, only object e_1 can be generated. Hence, some objects (or no object) from the set $\{r_1, r_2, \ldots, r_m\}$ can be generated in each membrane 2. The generated object r_j represents that the corresponding clause is satisfied by the variable assignment. In general, for the current variable x_i ($1 \leqslant i \leqslant n$), there are m clauses, which correspond to m rules in $R_{2,1,j}$ or $R_{3,1,j}$. For $R_{2,1,j}$ or $R_{3,1,j}$, the rules in which may complete at different steps. This process takes at most one RS-step. Finally, m copies of object e_1 can be generated in each membrane 2.

The application of rule $R_{4,1,j}$ starts if an object e_1 appears in a membrane 2. At that moment, object e_1 enters an arbitrary membrane with label $n+j$, and its polarization can be changed to positive. Similarly, this process also takes at most $2m$ RS-steps.

When the polarization of the membrane with label $n+1$ is changed to positive, rule $R_{5,1,1}$ (resp., $R_{6,1,1}$) enabled and applied, object $t_{1,1}$ (resp., $f_{1,1}$) enters the corresponding membrane. Then, by applying the rule $R_{7,1,1}$ (resp., $R_{8,1,1}$), object $t_{1,1}$ (resp., $f_{1,1}$) will be sent out of that membrane and object $t_{1,2}$ (resp., $f_{1,2}$) can be generated. Next, rules $R_{5,1,j}$ (resp., $R_{6,1,j}$) and $R_{7,1,j}$ (resp., $R_{8,1,j}$) can be applied one by one till object $t_{1,m+1}$ (resp., $f_{1,m+1}$) is generated in the membrane 2. When the object $t_{1,m+1}$ (resp., $f_{1,m+1}$) is generated, it means that application of all the rules in $R_{2,1,j}$ (resp., $R_{3,1,j}$), $R_{4,1,j}$, $R_{5,1,j}$ (resp., $R_{6,1,j}$) and $R_{7,1,j}$ (resp., $R_{8,1,j}$) have completed. At that moment, rule $R_{9,1,1}$(resp., $R_{10,1,1}$) is enabled and applied. The rules from $R_{5,1,1}$ (resp., $R_{6,1,1}$) to $R_{7,1,m}$ (resp., $R_{8,1,m}$) are used for checking whether or not the application of rules in $R_{2,1,j}$ and $R_{3,1,j}$ has completed. The rules from $R_{9,1,1}$ (resp., $R_{10,1,1}$) to $R_{11,1,m}$ (resp., $R_{12,1,m}$) are used for changing the polarization of membranes with label $n+j$ from positive to negative. Thus, the rule $R_{4,2,j}$ can be applied in the next iteration of variable x_1. Finally, the object $t_{1,2m+1}$ (resp., $f_{1,2m+1}$) can be generated in the membrane with label 2. At that moment, object $t_{1,2m+1}$ (resp., $f_{1,2m+1}$) will be sent out of that membrane. When the object $t_{1,2m+1}$ (resp., $f_{1,2m+1}$) appear in membrane 1, telling us that the application of rule $R_{13,1}$ (resp., $R_{14,1}$) completes, the application of rules $R_{15,1}$ starts. Under the influence of the promoter $t_{1,2m+1}$ and with the presence of object $f_{1,2m+1}$ the object $f_{1,2m+1}$ evolves to two copies of object a_2. At next step, each object a_2 enters the membrane with label 2, and its polarization can be changed to positive. Thus, the first iteration of variable x_i ($1 \leqslant i \leqslant n$) completes.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

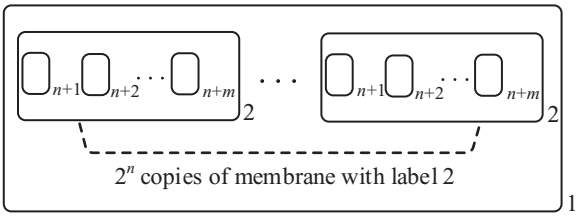


Fig. 1. The membrane structure at the moment when generation phase completes.

When the first iteration of variable x_i ($1 \leqslant i \leqslant n$) completes, the membranes with label 2 contain object a_2. Under the influence of the object a_2, rule $R_{1,2}$ can be applied in each membrane 2 simultaneously, and four membranes with label 2 can be generated. The following computing process is similar to the case of x_1. The system continues to assign values (*true* and *false*) to variables x_2, and all the satisfied clauses can be obtained. When the second iteration of variable x_i ($1 \leqslant i \leqslant n$) completes, the system continues to assign values (*true* and *false*) to variables $x_3, \ldots, x_n$. Similarly, every variable divides continuously, looking for all the clauses satisfied by the truth-assignment of variables. In general, for the ith iteration ($1 \leqslant i \leqslant n$) of variable x_i, when the execution of rule $R_{13,i}$ and $R_{14,i}$ finishes, object $f_{i,2m+1}$ evolves to two copies of a_{i+1} by applying rule $R_{15,i}$. Thus, the number of objects a_{i+1} is equal to the number of the membrane 2. But $t_{i,2m+1}$ and $f_{i,2m+1}$ may not be generated at the same time because the system works independently from the execution time of rules. For the ith iteration of variable x_i, it is obvious that $\sum_{1\leqslant j\leqslant m}\{e(R_{5,i,j}) + e(R_{7,i,j}) + e(R_{9,i,j}) + e(R_{11,i,j}) + e(R_{13,i,j})\}$ may not be equal to $\sum_{1\leqslant j\leqslant m}\{e(R_{6,i,j}) + e(R_{8,i,j}) + e(R_{10,i,j}) + e(R_{12,i,j}) + e(R_{14,i,j})\}$. The rule $R_{15,i}$ cannot be applied until objects $t_{1,2m+1}$ and $f_{1,2m+1}$ appear in the membrane 1. Hence, the rule $R_{15,i}$ has a synchronization function.

In general, for every membrane containing object $t_{i,1}$ (resp., $f_{i,1}$), in which the same rules are applied in each iteration of the loop. Hence, object $t_{i,2m+1}$ (resp., $f_{i,2m+1}$) in each membrane 2 can be generated simultaneously. Note that rule $R_{4,i,j}$ is applied in a non-deterministic way. Hence, the membranes to be applied from $n+1$ to $n+m$ are non-deterministically chosen. But it has no influence on the following result. With the execution of rule $R_{16,i}$, note that the polarization of membranes with label 2 can be changed from negative to positive. Hence, by applying the rule $R_{16,i}$, only one copy of object a_{i+1} will enter an arbitrary membrane 2. In addition, owing to the maximal parallelism of P systems, the application of the rule $R_{16,i}$ completes in each membrane 2 simultaneously. Hence, for each ith iteration ($1 \leqslant i \leqslant n$), it takes only one RS-step.

It should be pointed out that the system executes the above steps in an iteration manner till the assignments of all variables complete. In general, after $10mn+6n$ RS-steps, the generation phase will complete. At that moment, 2^n copies of membrane 2 can be generated (see Figure 1).

Checking phase:

When object a_{n+1} occurs in an arbitrary membrane 2, it means that the execution of all the aforementioned rules has completed. Note that the application of $R_{16,n}$ in each membrane follows the principle of maximal parallelism. Hence, objects a_{n+1} in each

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

membrane 2 will appear simultaneously. When generation phase finishes, there are some objects (or no object) from the set $\{r_1, r_2, \ldots, r_m\}$ in every membrane 2. These objects represent that the corresponding clause is satisfied by the assignment *true* (resp., *false*) of variable x_i. Thus, if one membrane contains all objects including $r_1, r_2, \ldots, r_m$, this means that the formula is satisfiable. With the appearance of an object a_{n+1} and an object r_1 (if this object exists) in a membranes 2, by using rule R_{17}, the object r_1 evolves to s_2. If object s_i $(2 \leqslant i \leqslant m)$ appears in a membrane 2, rule $R_{18,i}$ can be applied. If object s_m appears in a membrane 2, rule $R_{18,m}$ can be applied. Thus, object s_{m+1} can be generated. Please note that the rules are used in the time-free way. In general, it takes at most m RS-steps to complete the checking phase.

Output phase:

When checking phase completes, there will be the following two cases.

Affirmative answer: In this case, object s_{m+1} is generated in a membrane with label 2. It means that the formula is satisfiable. By applying rule R_{20}, Under the influence of promoter s_{m+1} in membrane 2, object a_{n+1} evolves to object $\mathcal{YES}$, and it can be sent out of that membrane by using the rule R_{21}. When the object $\mathcal{YES}$ appears in membrane 1, the application of rule R_{22} starts, object $\mathcal{YES}$ will be sent out of that membrane, and its polarization can be changed to negative. At step 1, the rule R_{19} is applied. When the execution of both rule R_{19} and R_{22} completes, by using the rule R_{23}, object $\mathcal{NO}$ in the environment enters membrane 1. Thus, when the system stops working, object $\mathcal{YES}$ will be in the environment. Therefore, we can draw a conclusion that there is at least one satisfiable solution. In general, the output phase takes at most 4 RS-steps.

Negative answer: In this case, object s_{m+1} does not appear in a membrane with label 2. Therefore, the rules from R_{20} to R_{23} cannot be applied. Eventually, when the system halts, object $\mathcal{NO}$ is still in the environment, we can draw a conclusion that there is no satisfiable solution. Hence, in this case, this process takes no RS-step.

5. An instance of $\Pi_{\mathcal{SAT}(m,n)}$

In this section, an instance is given to illustrate how $\Pi_{\mathcal{SAT}(m,n)}$ works. The feasibility and effectiveness of the proposed system is demonstrated by the instance as follows:

Example:

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_4).$$

As shown in Figure 2, it is the initial structure of $\Pi_{\mathcal{SAT}(m,n)}$. At step 1, rules $R_{1,1}$ and R_{19} can be applied at the same time. Rule $R_{1,1}$ can be applied to divide membrane 2 into new membranes with the same label. Simultaneously, object a_1 evolves to objects $t_{1,1}$ and $f_{1,1}$ in the two generated membranes. Then, by applying the rule $R_{2,1,j}$ (resp., $R_{3,1,j}$), under the influence of the promoter $t_{1,1}$ (resp., $f_{1,1}$) and with the presence of object $Y_{1,j}$ (resp., $N_{1,j}$) in membrane 2, object $Y_{1,j}$ (resp., $N_{1,j}$) evolves to object r_j and e_1. Other cases only object e_1 can be generated. When an object e_1 appears in membrane 2, the application of rule $R_{4,1,j}$ starts, and the object e_1 enters an arbitrary membrane with label $n+j$, and its polarization can be changed to positive. In the following figures, it should be noted

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

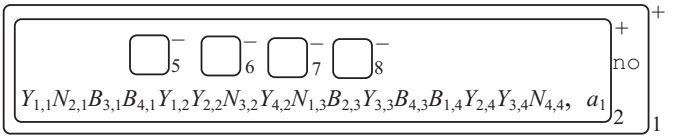


Fig. 2. The initial configuration.

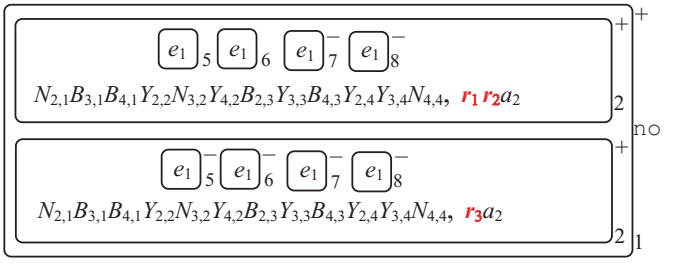


Fig. 3. After the execution of the first iteration.

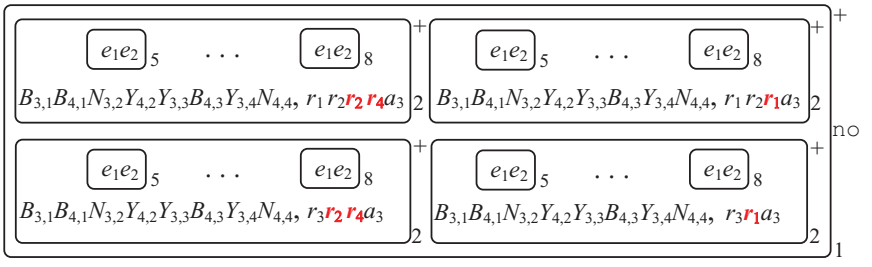


Fig. 4. After the execution of the second iteration.

that we will omit some substances that have no effect on the computing process and computing results.

When the polarization of the membrane with label from $n+1$ to $n+m$ is changed to positive, rules $R_{5,1,j}$ (resp., $R_{6,1,j}$) and $R_{7,1,j}$ (resp., $R_{8,1,j}$) can be applied one by one till object $t_{1,5}$ (resp., $f_{1,5}$) is generated in membrane 2. When an object $t_{1,5}$ (resp., $f_{1,5}$) appears in membrane 2, the rules $R_{9,1,j}$ (resp., $R_{10,1,j}$) and $R_{11,1,j}$ (resp., $R_{12,1,j}$) can be applied one by one till the object $t_{1,9}$ (resp., $f_{1,9}$) is generated in the membrane 2. Finally, under the influence of the promoter $t_{1,9}$ and with the presence of object $f_{1,9}$, object a_2 can be generated. At that moment, every object a_2 enters an arbitrary membrane 2, changing the polarization from negative to positive, and the application of rules for the first iteration completes. We can obtain the configuration (see Figure 3). It is uncertain whether the execution of rule R_{19} has completed. In Figure 3, however, we suppose that the execution of rule R_{19} has completed.

When the first iteration of variable x_i ($1 \leqslant i \leqslant n$) completes, the membranes with label 2 contain object a_2. Under the influence of object a_2, rule $R_{1,2}$ can be applied at the same time, by which four membranes with label 2 can be generated. When the application of rules for the second iteration completes, that is, when the application of rule $R_{16,3}$ finishes, we can obtain the configuration (see Figure 4). Similarly, when the execution of the third

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

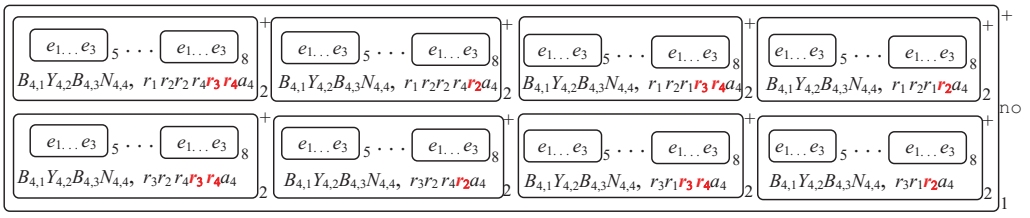


Fig. 5. After the execution of the third iteration.

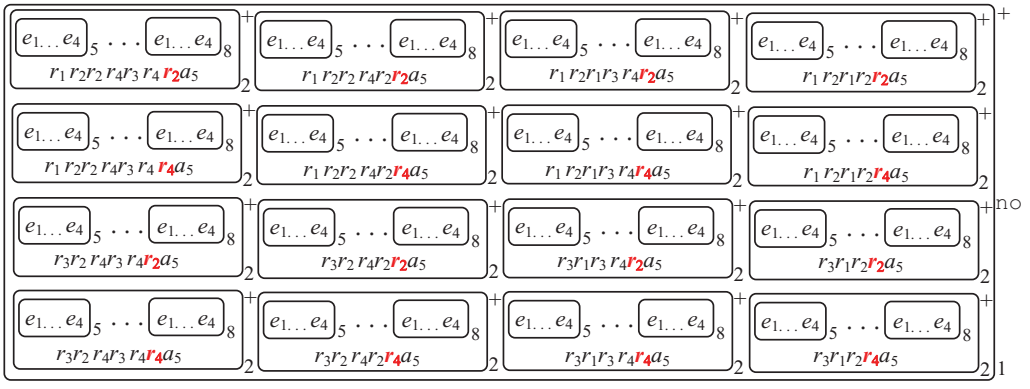


Fig. 6. After the execution of the fourth iteration.

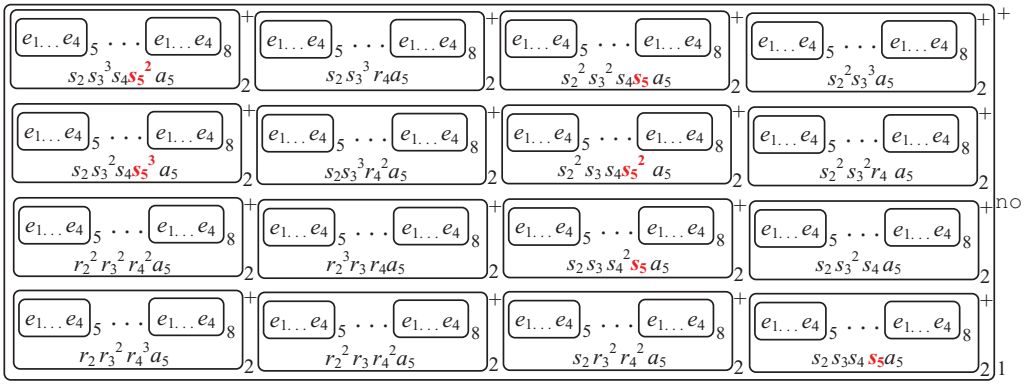


Fig. 7. After applying the rule R_{13} and $R_{14,j}$.

iteration completes, we can obtain the configuration (see Figure 5); when the execution of the fourth iteration completes, we can obtain the configuration (see Figure 6).

When object a_{n+1} (namely, a_5) occurs in a membrane 2, with the appearance object r_1 (if this object exists) in membranes 2, by using the rule R_{17}, the object r_1 can evolve to s_2. If the object s_i $(2 \leqslant i \leqslant m)$ appears in a membrane 2, rule $R_{18,j}$ can be applied. Finally, object s_{m+1} (namely, a_5) can be generated (see Figure 7).

When object a_5 presents in a membrane with label 2, rule R_{20} can be applied. By using the rule R_{20}, under the influence of promoter s_5, object a_5 evolves to object $\mathcal{YES}$. Eventually, the object $\mathcal{YES}$ is sent out of the membrane 1. Simultaneously, the object

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

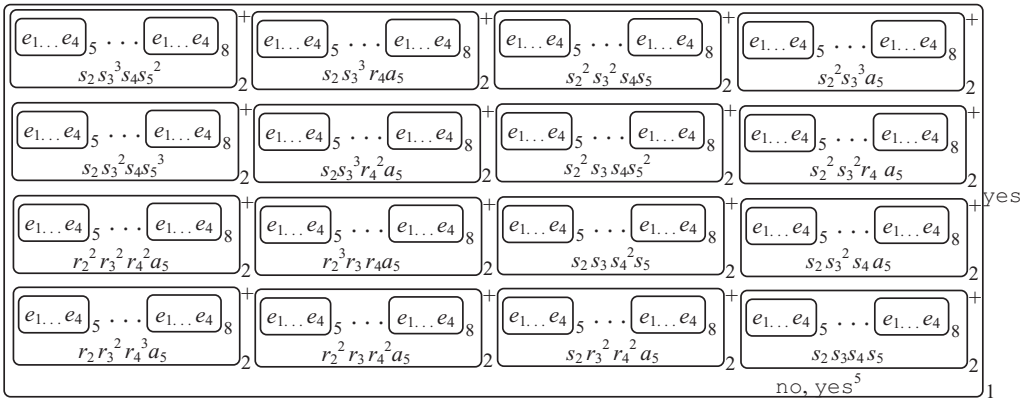


Fig. 8. The final configuration.

$\mathcal{NO}$ in the environment enters membrane 1. Thus, the system stops working. In this case, object $\mathcal{YES}$ will be in the environment, we can draw a conclusion that there is at least one satisfiable solution (see Figure 8). Hence, the formula is satisfiable.

6. Theoretical analysis of computational efficiency

1. Some formal details

   For a $\mathcal{SAT}$ instance with n Boolean variables and m clauses, the necessary resources to construct $\Pi_{\mathcal{SAT}(m,n)}$ can be drawn as follows:

   — Size of the set O : $7mn + 2m + 4n + 3$;
   — The number of initial membranes: $m + 2$;
   — Initial number of objects : 2;
   — The total number of rules: $15mn + m + 5n + 5$.

   When the computation halts, object $\mathcal{YES}$ (resp., $\mathcal{NO}$) appears in the environment if the formula γ is satisfiable (resp., not satisfiable). If the formula γ is satisfiable, the computation takes at most $10mn + m + 6n + 4$ RS-steps: generation phase takes at most $10mn + 6n$ RS-steps; checking phase takes at most m RS-steps; output phase takes at most 4 RS-steps. If the formula γ is not satisfiable, the computation takes at most $10mn + m + 6n - 1$ RS-steps (it takes no RS-step at the output phase). It is easy to prove that the system constructed above is time-free sound, time-free complete and time-free polynomially bounded. The $\mathcal{SAT}$ problem can be solved in polynomial RS-steps by a family of recognizer timed *AMPP* systems. Rules are applied in the maximally parallel manner. Furthermore, the application of rules can be regulated by promoters. Hence, the constructed P systems have high computational efficiency to solve the **NP**-hard problems even in the time-free manner.
2. Comparison with other time-free solution

   In Song et al. (2014, 2015b), the P systems are constructed in the semi-uniform way. In our work, the P systems are constructed in the uniform way. Our solution provides a general method for solving the SAT problem.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

Table 1. *Comparison of computing resource and RS-steps for the uniform way.*

Solution	The number of initial membranes	The total number of rules	Membrane polarizations	RS-step complexity(RS-steps)
Time-free uniform solution in Song et al. (2015a)	$m+n+2$	$2n^2+10mn+$ $4m+9n+5$	Three types	$2n^2+6mn+$ $3m+8n+4$
Uniform solution with timed *AMPP* systems	$m+2$	$15mn+m+$ $5n+5$	Two types	$10mn+m+$ $6n+4$

Compared with the constructed P systems in Song et al. (2015a), we only consider two types of membrane polarizations (+ and −). In our work, the constructed P with only two polarizations can still solve the $\mathcal{SAT}$ problem in polynomial RS-steps. For a P system, obviously, the following aspects are important to evaluate its computational efficiency:

— Less consumption of substances;
— Less rules and as few as possible of the maximal rule length;
— As simple as possible of the membrane structure;
— Less use of other computing resources, e.g., membrane polarizations;
— Less algorithm complexity, e.g., time complexity of algorithm. In timed P systems, we consider RS-steps of the system.

In this paper, we only consider P systems constructed with active membranes in a time-free manner and the constructed P systems work in a uniform way. In these cases, compared with the constructed P systems in Song et al. (2015a), the system constructed in our work requires fewer necessary resources and RS-steps (see Table 1).

7. Conclusions

Some biochemical reactions may be controlled by promoters in the active membranes. With the biological motivation, we proposed a new variant of timed P systems with active membranes. Under the influence of promoters, we proved that any Turing computable set of numbers can be generated in a time-free manner. Furthermore, we have constructed a uniform solution to the $\mathcal{SAT}$ problem in the framework of recognizer timed *AMPP* systems with rules of types (a), (b), (c), ($e^{(2)}$) by membrane division in polynomial time. The execution time of the involved rules has no influence on the correctness of the solution, which reflects the robustness of the biological system from the computational point of view. As a result, the constructed P systems have high computational efficiency to solve the **NP**-complete problems even in a time-free manner. Under the influence of promoters, the P systems require fewer necessary resources and RS-steps. Our work provides a new and effective solution to the $\mathcal{SAT}$ problem in a distributed and parallel manner.

In this paper, the $\mathcal{SAT}$ problem can be solved by membrane division for non-elementary membranes. It remains open whether the $\mathcal{SAT}$ problem can be solved by a uniform family of timed *AMPP* systems by membrane division for elementary membranes. In

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

addition, the polarizations of membranes in this kind of P systems are two types. As we know, promoters can guide the computing process. Therefore, under the influence of the promoters, it is of great interest to investigate the computational power by using *AMPP* systems with no polarizations and working in a time-free manner.

Recently, a different strategy of using rules, called flat maximal parallelism was proposed in Song et al. (2016c) and Pan et al. (2017). It is of interest to investigate the computational power of timed *AMPP* systems in a flat maximally parallel way.

This research is supported by the Educational Science Foundation of Chongqing China (KJ1601221), the Big Data Agricultural Research Center of Southeast Chongqing (2015XJPT02) and the National Natural Science Foundation of China (No. 61502063).

References

Alhazov, A., Pan, L. and Păun, G. (2004). Trading polarizations for labels in P systems with active membranes. *Acta Informatica* **41** (2-3) 111–144.

Bottoni, P., Martín-Vide, C. and Păun, G. (2002). Membrane systems with promoters/inhibitors. *Acta Informatica* **38** (10) 695–720.

Cavaliere, M. and Sburlan, D. (2005). *Time-Independent P Systems*. Lecture Notes in Computer Science 3365, Springer-Verlag, Berlin 239–258.

Gheorghe, M., Păun, G. and Pérez-Jiménez, M.J. (2013). Research frontiers of membrane computing: Open problems and research topics. *International Journal of Foundations of Computer Science* **24** (5) 547–623.

Guo, P., Ji, J. and Chen, H. (2015). Solving all-SAT problems by P systems. *Chinese Journal of Electronics* **24** (4) 744–749.

Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J. (2007). A uniform solution to SAT using membrane creation. *Theoretical Computer Science* **371** (1-2) 54–61.

Ionescu, M., Păun, G. and Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae* **71** (2) 279–308.

Leporati, A., Mauri, G., Zandron, C., Păun, G. and Pérez-Jiménez, M. J. (2009). Uniform solutions to SAT and subset sum by spiking neural P systems. *Natural Computing* **8** (4) 681–702.

Luo, Y., Xiong, Z., Tan, H. and Xia, S. (2015). A uniform algorithm of solving all-SAT using membrane systems. *Journal of Computational & Theoretical Nanoscience* **12** (12) 5825–5832.

Luo, Y., Xiong, Z., Chen, Y. and Tan, H. (2016). Efficient solutions to all-SAT by P systems with active membranes. *Journal of Computational & Theoretical Nanoscience* **13** (6) 3887–3894.

Minsky, M. (1967). *Computation: Finite and Infinite Machines*, Prentice-Hall Inc.

Pan, L., Zeng, X. and Zhang, X. (2011). Time-free spiking neural P systems. *Neural Computation* **23** (5) 1320–1342.

Pan, L. and Pérez-Jiménez, M.J. (2010). Computational complexity of tissue-like P systems. *Journal of Complexity* **26** (3) 296–315.

Pan, L., Wang, Y.S., Jiang, S. and Song, B. (2017). Flat maximal parallelism in tissue P systems with promoters. *Romanian Journal of Information Science and Technology* **20** (1) 42–56.

Păun, A. and Păun, G. (2002). The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (3) 295–305.

Păun, A. and Păun, G. (2007). Small universal spiking neural P systems. *Biosystems* **90** (1) 48–60.

Păun, G. (1999a). Introduction to membrane computing. *Theoretical Computer Science* **287** (1) 73–100.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press

Păun, G. (1999b). P systems with active membranes: Attacking NP complete problems. *Journal of Automata Languages & Combinatorics* **6** (1) 75–90.

Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences* **61** (1) 108–143.

Păun, G., Pérez-Jiménez, M.J. and Riscos, A. (2008). Tissue P systems with cell division. *International Journal of Computers Communications & Control* **3** (3) 295–303.

Păun, G., Suzuki, Y., Tanaka, H. and Yokomori, T. (2004). On the power of membrane division in P systems. *Theoretical Computer Science* **324** (1) 61–85.

Pérez-Jiménez, M.J. and Sosik, P. (2015). An optimal frontier of the efficiency of tissue P systems with cell separation. *Fundamenta Informaticae* **138** (1-2) 45–60.

Song, B. and Pan, L. (2015a). Computational efficiency and universality of timed P systems with active membranes. *Theoretical Computer Science* **567**, 74–86.

Song, B., Song, T. and Pan, L. (2015b). Time-free solution to SAT problem by P systems with active membranes and standard cell division rules. *Natural Computing* **14** (4) 673–681.

Song, B., Pérez-Jiménez M.J. and Pan, L. (2015c). Efficient solutions to hard computational problems by P systems with symport/antiport rules and membrane division. *BioSystems* **130**, 51–58.

Song, B., Zhang, C. and Pan, L. (2016a). Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences* **378**, 177–193.

Song, B., Pérez-Jiménez M.J. and Pan, L. (2016b). An efficient time-free solution to SAT problem by P systems with proteins on membranes. *Journal of Computer and System Sciences* **82** (6) 1090–1099.

Song, B., Pérez-Jiménez M.J. and Pan, L. (2017a). An efficient time-free solution to QSAT problem using P systems with proteins on membranes. *Information and Computation* **256C**, 287–299.

Song, B., Song, T. and Pan, L. (2017b). A time-free uniform solution to subset sum problem by tissue P systems with cell division. *Mathematical Structures in Computer Science* **27**, 17–32.

Song, B., Pérez-Jiménez M.J., Păun, G. and Pan, L. (2016c). Tissue P systems with channel states working in the flat maximally parallel way. *IEEE Transactions on NanoBioscience* **15** (7) 645–656.

Song T., Macias-Ramos, L. and Pan, L. (2014). Time-free solution to SAT problem using P systems with active membranes. *Theoretical Computer Science* **529**, 61–68.

Song, T. and Pan, L. (2016). Spiking neural P systems with request rules. *Neurocomputing* **193**, 193–200.

Wu, T., Zhang, Z. and Pan, L. (2016). On languages generated by cell-like spiking neural P systems. *IEEE Transactions on Nanobioscience* **15** (5) 455–467.

Zeng, X., Xu, L., Liu, X. and Pan, L. (2014a). On languages generated by spiking neural P systems with weights. *Information Sciences* **278** (10) 423–433.

Zeng, X., Pan, L. and Pérez-Jiménez, M.J. (2014b). Small universal simple spiking neural P systems with weights. *Science China* **57** (9) 1–11.

Zhang, G., Gheorghe, M. and Li, Y. (2012). A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. *Natural Computing* **11** (4) 701–717.

Zhang, G., Gheorghe, M., Pan, L. and Pérez-Jiménez, M.J. (2014). Evolutionary membrane computing: A comprehensive survey and new results. *Information Sciences* **279**) 528–551.

Zhang, X., Liu, Y., Luo, B. and Pan, L. (2014). Computational power of tissue P systems for generating control languages. *Information Sciences* **278** (10) 285–297.

Zhang, X., Pan, L. and Păun, A. (2017). On the universality of axon P systems. *IEEE Transactions on Neural Networks & Learning Systems* **26** (11) 2816–2829.

https://doi.org/10.1017/S0960129518000282 Published online by Cambridge University Press