IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 April 2023, accepted 18 May 2023, date of publication 29 May 2023, date of current version 7 June 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3280801*

RESEARCH ARTICLE

# Flip-Flop P Systems With Proteins on Membranes in a Time-Free Manner

**XIAOMING WAN[1], YI LIU[2], AND YUEGUO LUO[3]**
[1]College of Electronics and IoT Engineering, Chongqing Industry Polytechnic College, Chongqing 401120, China
[2]College of Rail Transit and Aviation Services, Chongqing Industry Polytechnic College, Chongqing 401120, China
[3]College of Big Data and Intelligent Engineering, Yangtze Normal University, Chongqing 408100, China

Corresponding author: Yueguo Luo (ygluo@yznu.edu.cn)

This work was supported by the Science and Technology Research Program of Chongqing Municipal Education Commission under Grant KJZD-K202003201.

**ABSTRACT** Flip-flop P systems with proteins are a bio-inspired variant of cell-like P systems in membrane computing, where proteins can control the execution of rules. In this work, firstly, in order to simulate the fact that the execution time of biochemical reactions is uncertain, considering time-freeness, we therefore construct a novel variant, namely timed flip-flop P systems with proteins, where the protein on each membrane only has two types of working states, and such a system runs under the time-freeness mode; secondly, we study the computation power of this variant, and it is shown that a system with only one membrane and a maximum rule length of 4 is Turing universal; moreover, based on the variant, a solution to the $\mathcal{SAT}$ problem is obtained by the constructed system in polynomial time. Our work indicates that the constructed variant with time-freeness can still solve the $\mathcal{SAT}$ problem in feasible time. Because time-freeness of rules is employed, the variant may be more suitable for particular applications.

**INDEX TERMS** P system, protein, university, $\mathcal{SAT}$, time-freeness.

## I. INTRODUCTION

Membrane computing, which is inspired from processing of energy and information in biological cells, was conceived by G. Păun in 1998. The models based on membrane computing are called P systems, and the official article emerged in 2000 [1]. Living cells are of small sizes and strong self-repair abilities. In theory, a computing device based on biological reality also has many desirable characteristics, such as small size, excellent reliability, strong fault tolerance, and outstanding parallel computing capability. Computing based on biological reality can provide new tools for manipulating information and simulating biological systems, which leads many scholars to explore this field. By applying DNA molecules and some biochemical reaction operations, Professor Adleman successfully solved the Hamiltonian path problem with 7 vertices in a tube [2]. Recently, inspired by spiking neural systems, Kaushik Roy and other scientists published important research results in *Nature* [3], proposing a new type of neural computing system from the perspective of biological computing, and studied the algorithm design and hardware realization of the system. Currently, membrane computing is attracting widespread attention from scholars in various fields, including computer scientists, biologists, and mathematicians. Currently, a variety of membrane systems are Turing universal [4], [5], [6], [7], [8], [9]. Relative to computational efficiency, various variants are applied to solve **NP**-hard problems, such as the $\mathcal{SAT}$ problem [10], [11], vertex cover problem [12] and the 3-coloring problem [13], [14]. In addition, arithmetic operations [15], [16] and logical expressions [17] have been solved theoretically. Inspired by biochemical reactions, a number of variants have been proposed [18], [19], [20], [21]. Moreover, optimization problems [22], [23], fault diagnosis [24] and artificial intelligence [25] can also be deployed by P systems. Additionally, one can refer to recently published books on real-life applications [26], [27]. More recent research results and information of this area can be viewed on the website http://ppage.psystems.eu/, and one can refer to recent review articles (e.g., [28]).

The associate editor coordinating the review of this manuscript and approving it for publication was Rajeswari Sundararajan.

It is known that biochemical reactions may be controlled by proteins on membranes and therefore A. Păun proposed a novel variant, namely P systems with proteins on membranes ($\mathcal{PPM}$ systems, for short) [29]; where, proteins are placed on membranes, and objects may exist in corresponding membranes. Notably, proteins can be applied to govern substance evolution, which coincides with the biological fact from the biological point of view. In addition, importantly, $\mathcal{PPM}$ systems use only one copy of protein and object when a rule is applied. Although the variant has been proposed several years ago, the existing research results of $\mathcal{PPM}$ systems are still relatively limited. In [30], the system constructed with one membrane was proved computationally universal. Additionally, when division rule is applied, such P systems can solve the $\mathcal{SAT}$ problem [31], [32]. In [33], proteins were introduced into active membranes, where rules are guided by proteins and polarizations. However, relative to these results, the application of rules being used must stop running in a unit time. Nevertheless, biological system is highly robust. Hence, it is necessary to build membrane systems with fault tolerance. Namely, the constructed P systems can work independently from execution time of rules. As far as time-free is concerned, the approach was proposed in [34]. Recently, relative to **NP**-complete problems, some results have been obtained with time-freeness [35], [36].

In [37], time-freeness was introduced into $\mathcal{PPM}$ systems; nevertheless, the protein on a membrane may have a variety of working states; that is, there is no restriction on the number of proteins. With further study of $\mathcal{PPM}$, there may be a variety of proteins on a membrane; however, for each protein, the working states of each protein are finite. A noteworthy feature of such system is the structure of cell-like P systems [1]. Based on the biological reality, some scholars have proposed flip-flop P systems with proteins on membranes ($\mathcal{FPPM}$, for short), where there are only two working states for each protein, e.g., from $p$ to $p'$ or back. Therefore, its working mode is similar to a trigger. In [38] and [39], its computational universality was studied; however, under time-freeness, the research on computational property of $\mathcal{FPPM}$ has not been involved; hence, in our work, we will introduce time-freeness into $\mathcal{FPPM}$, namely, timed flip-flop P systems with proteins on membranes ($\mathcal{TFPPM}$ systems, for short), thereby constructing a robust computational system that would not be influenced by the execution time of rules.

The main contributions of this article are the following aspects.

(i) A novel variant ($\mathcal{TFPPM}$ systems) is established, where the computing result is irrelevant to the execution time of each rule. Hence, this variant indicates the robustness of biochemical reactions.

(ii) The university is explored for generating numbers. We obtain the result that only one membrane and a maximum rule length of 4 is Turing universal even running with time-freeness and limited rule types.

(iii) The $\mathcal{SAT}$ problem is considered to explore its computational efficiency. In our research, owing to membrane division, we obtained the uniform solution of an **NP**-complete problem.

(iv) The variant can better reflect the uncontrollability of rules execution time, thus a more robust system is constructed; therefore, in the sense that this variant may be more suitable for particular applications.

This paper will be written as the following structure. First, we introduce some foundations related to this article. In section III, the variant $\mathcal{TFPPM}$ is established. Next, the computational power of the variant is explored to generate numbers. In section V, $\mathcal{TFPPM}$ is employed to solve the $\mathcal{SAT}$; particularly, the efficiency and feasibility of $\mathcal{TFPPM}$ are demonstrated by one example in section VI. In the end, some conclusions are presented.

## II. FOUNDATIONS

### A. FORMAL LANGUAGE THEORY

An alphabet $O$ is an infinite non-empty set in which the elements are symbols. $O^*$ denotes the set of strings composed by symbols in $O$ or the empty string $\lambda$ (without a symbol in it), and $O^+$ denotes the set of strings composed by the symbol in $O$ but without $\lambda$, namely, $O^+ = O^* - \lambda$. $O^*$ and $O^+$ can contain an infinite variety of sets composed by symbols from $O$. Given a string $w$ ($w \in O^*$), the length of $w$, which is the quantity of symbols in this string, denoted by $|w|$. Note that if a symbol in a string appears multiple times, the repeated symbols needs to be counted in the length of the string. For example, relative to a string $a^m b^n c$, its length is $m + n + 1$. For an empty string, obviously, $|\lambda| = 0$. Two strings can be concatenated, and the result is a sequence of strings composed by the two strings; for example, if strings $w$ and $u$ are concatenated, the string $wu$ would be the result of concatenation.

A multiset from an alphabet $O$ is represented by $(O, f)$. If $O = \{u_1, \ldots, u_m\}, 1 \leq i \leq m$, $f(u_i)$ is the multiplicity of object $u_i$; namely, $f(u_i)$ is the quantity of $u_i$. If $O = \{u_1, \ldots, u_m\}$, the multiset $(O, f)$ is represented by $u_1^{f(u_1)} u_2^{f(u_2)}) \ldots u_m^{f(u_m)}$.

### B. REGISTER MACHINES

*Definition 1:* A register machine is a five tuple, namely $M = (m, H, l_0, l_h, I)$, where, $m$ is registers, $H$ denotes a labels set, $l_0$ (resp., $l_h$) $\in H$ denotes the initial instruction (resp., halting instruction), $I$ represents instructions of three categories:

- $l_i : (\mathcal{ADD}(r), l_j, l_k)$ (register machine $r$ can be added with 1, and $l_j$ or $l_k$ can be performed);
- $l_i : (\mathcal{SUB}(r), l_j, l_k)$ (the register would be subtracted by 1 if its value is non-zero, and $l_j$ can be used; if not, $l_k$ can be used);
- $l_h : \mathcal{HALT}$.

Initially, a system runs the instruction with label $l_0$, and all register machines are empty. Subsequently, instructions are activated automatically and continuously until $l_h$ is used,

and then the system stops running. Like this, a number is generated in register 1; notably, *NRE* represents that a system can generate all set of Turing computable numbers.

### C. TIMED P SYSTEMS

Relative to membrane computing, in the past each rule of most variants is assumed to have an identical unit; namely, all rules are assumed to have a unit time from the beginning to the end of rules execution. However, relative to time-freeness, the execution time of an arbitrary rule is uncertain, thereby reflecting uncontrollable time in biochemical reactions, which allows the system to break the time limitation and thus has better robustness. When such a system is running, a global clock is assumed to contain a sequence of equal time segments from time 0. When there exists an executable rule, this moment corresponds to a RS-step, and we call this moment step 1. However, it is important that the execution time is not determinable. In a $\mathcal{TFPPM}$, $e$ denotes that $\mathcal{TFPPM}$ systems run with the working mode of time-freeness; $e(R) \rightarrow \mathbb{N}$ denotes that the execution time is mapped to a natural number. When a rule $R$ is executed at time $t$, this moment corresponds to a RS-step, and rule $R$ stops operation at time $t + e(R)$, where objects in the rule would be executed by other rules at time $t + e(R) + 1$.

## III. $\mathcal{TFPPM}$ SYSTEMS

In this section, based on the robustness of biochemical reactions, we introduce time-freeness into $\mathcal{FPPM}$, thus construct a new variant, namely timed flip-flop P systems with proteins, where the computing result is irrelevant to the execution time of each rule; additionally, proteins on each membrane only has two types of working states.

*Definition 1:* A $\mathcal{TFPPM}$ system (degree $m \geq 1$) is the following tuple:

$$\Pi = (O, P, \mu, E, \alpha_1/\beta_1, \ldots, \alpha_m/\beta_m, R, e, i_{out}),$$

where

- $O$ is an alphabet of objects;
- $P$ represents non-empty alphabets of proteins ($O \cap P = \varnothing$). The protein on a membrane only has two working states.
- $E$ represents an infinite number of objects in the environment;
- $\mu$ represents nested membrane structure with $m$ nodes;
- $\alpha_i \subseteq O(1 \leq i \leq m)$, are multisets located in membrane $i$;
- $\beta_i \subseteq P(1 \leq i \leq m)$, are objects of proteins located on membrane $i$;
- $e$ denotes that $\mathcal{TFPPM}$ systems run with the working mode of time-freeness;
- $i_{out}$ is the output region ($i_{out} \in \{0, 1, \ldots, m\}$);
- $R$ represents rules associated with a membrane label, where a protein $p$ on a membrane $h$ is denoted by $[p|\ ]_h$.

(i) evolution rules:

(a) $[p|u]_h \rightarrow [p'|v]_h, u, v \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
When an object occurs in a membrane, it is evolved to another object. Applied such a rule, the protein may change or not change.

(b) $u[p|\ ]_h \rightarrow v[p'|\ ]_h, u, v \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
When an object occurs outside a membrane, it evolved to another object. Applied such a rule, the protein may change or not change.

(c) $[p|u]_h \rightarrow v[p'|\ ]_h, u, v \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
When an object occurs in a membrane, it is evolved to another object and comes out from that membrane. Applied such a rule, the protein may change or not change.

(d) $u[p|\ ]_h \rightarrow [p'|v]_h, u, v \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
When an object occurs outside a membrane, it enters the membrane and can be evolved to another object. Applied such a rule, the protein may change or not change.

(e) $u[p|v]_h \rightarrow w[p'|z]_h, u, v, w, z \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
When an object occurs outside a membrane and one object occurs inside this membrane, the outside object enters the membrane, while the inside object comes to the outside region. Applied such a rule, each object in the rule may be evolved to another object, and protein may change or not change.

(ii) division rules:

(f) $[p|u]_h \rightarrow [p'|v]_h[p'|w]_h, u, v, w \in O, p, p' \in P$, $h \in \{1, \ldots, m\}$.
The initial membrane may be an elementary membrane or non-elementary membrane. When applied such a rule, the protein on the initial membrane may be changed; simultaneously, $v$ and $w$ appear in the new membranes.

Initially, the system is denoted by

$$(\alpha_1/\beta_1, \ldots, \alpha_m/\beta_m, \mu),$$

that is, $\alpha_1, \ldots, \alpha_m$ (resp., $\beta_1, \ldots, \beta_m$) are placed in (resp., on) the corresponding membranes. At each step, the configuration is described by $\mu$ and corresponding objects including proteins. For each evolution rule, we define its rule length with the quantity of objects located in each region associated with this rule. with non-deterministic maximally parallel strategy [1], the transitions can be obtained. Finally, when no rules are available and no rules are being executed, the system reaches completion.

When a $\mathcal{TFPPM}$ is running, the computing result is irrelevant to the execution time of rules. For example, a $\mathcal{TFPPM}$ has the structure $\mu = [[\ ]_2]_1$; a multiset of objects $ab$ exists in membrane 2, and protein $D$ (resp., $E$) exists on membrane 1 (resp., membrane 2); in addition, there are three rules associated with the membranes:

$$R_1 \equiv [E|a]_2 \rightarrow u[E|\ ]_2.$$

$$R_2 \equiv [E|b]_2 \rightarrow u[E|\,]_2.$$
$$R_3 \equiv [D|u]_1 \rightarrow w[D'|\,]_1.$$

A $\mathcal{TFPPM}$ and $\mathcal{FPPM}$ may run differently. In Figure 1, an example will illustrate the difference between these two approaches. First, we consider the execution of $\mathcal{FPPM}$. Initially, rules $R_1$ and $R_2$ are activated simultaneously, and two copies of object $u$ will generated in membrane 1. Next step, rule $R_3$ is available. Owing to maximal parallelism, rule $R_3$ can be executed multiple times, therefore two copies of object $w$ come to the environment as the computing result.

Next, we consider the strategy of time-freeness to run a $\mathcal{TFPPM}$. Initially, rules $R_1$ and $R_2$ are activated simultaneously; however, these rules may have different execution time, namely, $e(R_1) \neq e(R_2)$. Hence, two copies of object $u$ will be produced successively. When one copy of object $u$ is produced, rule $R_3$ is available, and object $w$ comes to the environment, replacing the protein $E$ by $E'$. In this case, because $R_3$ has already changed the protein on membrane 1, the rule will no longer be used. Hence, rule $R_3$ can be executed once, therefore an object $w$ will be transferred to outside of the skin membrane.

Therefore, the computing process including computing result of a $\mathcal{TFPPM}$ and $\mathcal{FPPM}$ may be different.

*Definition 2:* a recognizer $\mathcal{TFPPM}$ system is defined as follows:

$$\Pi = (O, P, \Sigma, \mu, E, \alpha_1/\beta_1, \ldots, \alpha_m/\beta_m, R, e, i_{in}, i_{out}),$$

where

- $\Sigma$ is an input alphabet from set $O$;
- $\mathcal{YES}, \mathcal{NO} \in O$;
- $e$ means that the system runs with time-freeness;
- $i_{in}$ (resp., $i_{out}$) represents the input (resp., output) region;
- The working alphabet is composed of object $\mathcal{YES}$ and $\mathcal{NO}$;
- Computations associated with the $\mathcal{TFPPM}$ system will halt;
- When $\mathcal{TFPPM}$ system reaches completion, the region $i_{out}$ would generate $\mathcal{YES}$ or $\mathcal{NO}$.

The other parameters are defined as in Definition 1. At the initial configuration, the input multiset $i_{in}$ appears in a membrane. A recognizer $\mathcal{TFPPM}$ system uses maximum parallelism as the strategy for applying rules. Furthermore, rules are applied non-deterministically. Finally, a recognizer $\mathcal{TFPPM}$ reaches the halting computation with certainty and produces related objects, which are stored in $i_{out}$. It must be emphasized that $e$ denotes time-freeness, meaning that operation time associated to rules from start to finish cannot be determined. When the system reaches completion, a result would appear in the region $i_{out}$. If $\mathcal{YES}$ (resp., $\mathcal{NO}$) appears, we call it an accepting (resp., rejecting) computation.

*Definition 3:* $X = (I_X, \theta_X)$ denotes a decision problem, where $I_X$ is instances, and $\theta_X$ represents a predicate of the instances. The problem can be solved in polynomial time, if the following holds:

(i) $\Pi$ is polynomially uniform by Turing machines;

(ii) Relative to a $I_X$, there is a pair $(cod, s)$ of polynomial-time computable functions such that:

- Suppose $u$ corresponds to an instance, $u \in I_X$, $s(u)$ is a natural number; additionally, $cod(u)$ represents an input multiset of $\mathcal{TFPPM}$ system.
- Relative to $(X, cod, s)$, such a system is complete with time-freeness. Suppose $u \in I_X$ relative to a problem with time-freeness, computations of $\Pi(s(u), e)$ with $cod(u)$ is an accepting one.
- Relative to $(X, cod, s)$, such a system is sound with time-freeness. With regard to $u \in I_X$, $\mathcal{TFPPM}$ system has an accepting computation, $\theta_X(u) = 1$;
- Relative to $(X, cod, s)$, such a system is polynomially bounded with time-freeness. Notably, the computing result is therefore irrelevant to the execution time of rules, and we have the polynomial function $p(n)$ such that for each $u \in I_X$; hence, $\mathcal{TFPPM}$ system must stop computation after $p(|u|)$ RS-steps ($p$ is a polynomial function).

*Definition 4:* The maximum rule length of a $\mathcal{TFPPM}$ system is equivalent to that of evolution rules in the system. $PMC^f_{\mathcal{TFPPM}(k)}$ indicates that a family of recognizer $\mathcal{TFPPM}$ systems can obtain a uniform solution to the class of decision problems in polynomial time, where, $k$ represents the maximum length in the $\mathcal{TFPPM}$, $f$ represents the time-free mode.

*Definition 5:* $NOP^f_m(rule_k)$ is the set of natural numbers generated by $\mathcal{TFPPM}$ systems, where $m$ is the number of membranes and $f$ represents the time-free mode; moreover, *rule* indicates rule types, e.g., types from (a) to (f). and $k$ is the maximal length of the corresponding rules.

## IV. UNIVERSITY OF $\mathcal{TFPPM}$ SYSTEMS

$M = (m, H, l_0, l_h, I)$ denotes the register machine with $m$ registers. When a register is used to the device of generate numbers, ADD instruction would not be used on register 1, and the register 1 stores the generated numbers. Once the system reaches completion, all registers except register 1 are empty.

For details of automata theory, one can refer to [40].

*Theorem 1:* $NOP^f_1((c)_2, (d)_2, (e)_4) = NRE$.

*Proof:* A $\mathcal{TFPPM}$ system is designed as follows.

$$\Pi = (O, P, \Sigma, \mu, \varnothing, w_1/z_1, R, e, i_{out}),$$

where

- $O = \{l, l', l'', l''', l^{iv}, l^{v} | l \in H\} \cup \{a_r | 1 \leq r \leq m\}$;
- $P = \{p, p'\}$;
- $E = \{l'' | l \in H\} \cup \{a_r | 1 \leq r \leq m\}$;
- $w_1 = \{l_0\}$;
- $\mu = [\,]_1$;
- $i_{out} = 1$.

Object $a_r$ will locate in membrane 1, which number can be viewed as the value of register $r$; when object $l_h$ exists in this membrane, the system reaches completion, and the

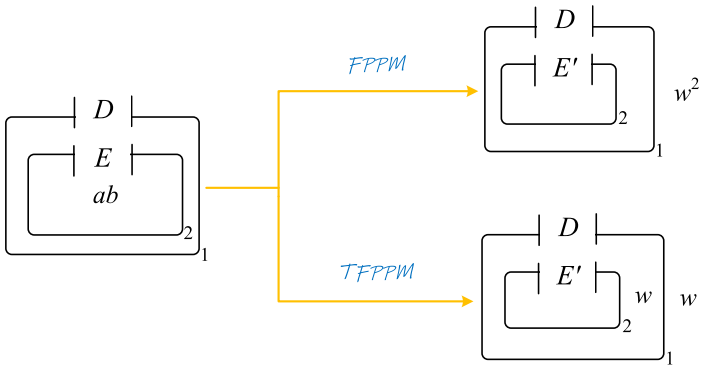



**FIGURE 1.** The computing results of a $\mathcal{TFPPM}$ and $\mathcal{FPPM}$.

value is the computation result. $l_i$ : $(\mathcal{ADD}(r), l_j, l_k)$ (resp., $l_i$ : $(\mathcal{SUB}(r), l_j, l_k)$) corresponds to the ADD instruction (resp., SUB instruction).

### A. RULES OF ADD INSTRUCTION

$$R_1 \equiv a_r[p|l_i]_1 \to l_i'[p|a_r]_1.$$
$$R_2 \equiv l_i'[p|\,]_1 \to [p|l_j]_1.$$
$$R_3 \equiv l_i'[p|\,]_1 \to [p|l_k]_1.$$

The ADD instruction works as follows. At a certain RS-step, the system starts to apply rule $R_1$; thus, the outside object $a_r$ enters membrane 1, and the inside object $l_i$ comes to the outside region to start simulating ADD instruction. This computing process will generate the object $a_r$ in membrane 1. Next, $R_2$ or $R_3$ can be applied non-deterministically, and $l_j$ or $l_k$ would appear in the membrane.

### B. RULES OF SUB INSTRUCTION

$$R_4 \equiv l_i''[p|l_i]_1 \to l_i'''[p'|l_i']_1.$$
$$R_5 \equiv l_i'''[p'|a_r]_1 \to a_r[p|l_i''']_1.$$
$$R_6 \equiv [p'|l_i']_1 \to l_i^{iv}[p|\,]_1.$$
$$R_7 \equiv l_i'''[p|\,]_1 \to [p'|l_i^{v}]_1.$$
$$R_8 \equiv l_i^{iv}[p|l_i''']_1 \to l_i''[p|l_j]_1.$$
$$R_9 \equiv l_i^{iv}[p'|l_i^{v}]_1 \to l_i''[p|l_k]_1.$$

At a certain RS-step, the system starts to execute rule $R_4$, thus the state of $p$ is changed to $p'$, and $l_i'$ including $l'''$ appear in corresponding regions. Next, two cases would occur in membrane 1.

(i) Object $a_r$ exists in the membrane. At the second RS-step, $R_5$ and $R_6$ can be applied simultaneously. Because of the protein $p'$ and object $l_i'''$, object $a_r$ leaves membrane 1 by applying rule $R_5$; simultaneously, rule $R_6$ is available, and $l_i'$ comes to the environment and is changed to $l_i^{iv}$. Notably, the execution of $R_5$ and $R_6$ may stop at the different time because of time-freeness; however, note that when $l_i'''$ and $l_i^{iv}$ appear in the corresponding regions, the next rule $R_8$ would be used; hence, only both $R_5$ and $R_6$ reach completion, rule $R_8$ can be used, thus object $l_j$ appears in the region, and protein $p$ replaces $p'$. Hence, it takes 3 RS-steps. In this case, applied rules and corresponding objects including proteins are shown in Table 1. Overall, rules are activated successively as follows.

$$R_4 \to \{R_5, R_6\} \to R_8$$

(ii) Object $a_r$ does not exist in the membrane. Next, only $R_6$ can be applied, thus $l_i'$ comes to the environment and is changed to $l_i^{iv}$. Subsequently, $R_7$ can be applied, the protein $p'$ will replace $p$; simultaneously, $l_i'''$ comes to membrane 1 and is changed to $l_i^{iv}$. Finally, because of the protein $p'$ and objects $l_i^{iv}$ including $l_i^{v}$, rule $R_9$ is available, thus object $l_k$ appears in the membrane, and protein $p$ generates for the purpose of simulating the next instruction. Therefore, the procedure above can correctly simulate instruction $l_k$. Evidently, it takes 4 RS-steps. In this case, applied rules and corresponding objects including proteins are shown in Table 2. Overall, rules are activated successively as follows.

$$R_4 \to R_6 \to R_7 \to R_9$$

As we have mentioned, the constructed system can work independently from execution time of rules. Finally, object $l_h$ would be generated in membrane 1, which indicates the system reaches completion.

## V. A UNIFORM SOLUTION TO THE $\mathcal{SAT}$ PROBLEM BASED ON $\mathcal{TFPPM}$ SYSTEMS

### A. CONSTRUCTING $\mathcal{TFPPM}$ SYSTEMS TO SOLVE THE $\mathcal{SAT}$

*Theorem 2:* $(\mathtt{SAT} \in PMC^{f}_{\mathcal{TFPPM}(4)})$:

Proof: A SAT with $n$ Boolean variables and $m$ clauses is the following formula:

$$C_j = y_{1,j} \vee \cdots \vee y_{p_j,j},$$

where $y_{i,j} \in \{x_l,\ \neg x_l | 1 \le l \le n\}$, $1 \le i \le p_j$, $1 \le j \le m$; $\neg x_l$ is the negation of a propositional variable $x_l$.

we encode a formula $\gamma$ by $cod(\gamma)$ as follows:

$$cod(\gamma) = B_{1,1} \cdots B_{n,1} B_{1,2} \cdots B_{n,2} \cdots B_{1,m} \cdots B_{n,m},$$

**TABLE 1.** computations if object $a_r$ exists in cell 1.

| RS-steps | applying rules | $E$ (do not contain $a_r$) | $P$ | $w_1$ |
|---|---|---|---|---|
| 0 | no rule | $l_i''$ | $p$ | instructions to be applied, e.g., $l_0$ |
| 1 | $R_4 : l_i''[p\|l_i]_1 \to l_i'''[p'\|l_i']_1$ | $l_i'''$ | $p'$ | $l_i'$ |
| 2 | $R_5 : l_i'''[p'\|a_r]_1 \to a_r[p\|l_i''']_1$ | $l_i^{iv}$ | $p$ | $l_i'''$ |
| | $R_6 : [p'\|l_i']_1 \to l_i^{iv}[p\|\,]_1$ | | | |
| 3 | $R_8 : l_i^{iv}[p\|l_i''']_1 \to l_i''[p\|l_j]_1$ | $l_i''$ | $p$ | $l_j$ |

**TABLE 2.** computations if object $a_r$ do not exist in cell 1.

| RS-steps | applying rules | $E$ (do not contain $a_r$) | $P$ | $w_1$ |
|---|---|---|---|---|
| 0 | no rule | $l_i''$ | $p$ | instructions to be applied, e.g., $l_0$ |
| 1 | $R_4 : l_i''[p\|l_i]_1 \to l_i'''[p'\|l_i']_1$ | $l_i'''$ | $p'$ | $l_i'$ |
| 2 | $R_6 : [p'\|l_i']_1 \to l_i^{iv}[p\|\,]_1$ | $l_i'''l_i^{iv}$ | $p$ | $\varnothing$ |
| 3 | $R_7 : l_i'''[p\|\,]_1 \to [p'\|l_i^v]_1$ | $l_i^{iv}$ | $p'$ | $l_i^v$ |
| 4 | $R_9 : l_i^{iv}[p'\|l_i^v]_1 \to l_i''[p\|l_k]_1$ | $l_i''$ | $p$ | $l_k$ |

where, $B_{i,j}$ is denoted by the following multisets:

$$B_{i,j} = \begin{cases} D_{i,j} : x_i \text{ is in } C_j; \\ E_{i,j} : \neg x_i \text{ is in } C_j; \\ F_{i,j} : \text{neither } x_i \text{ nor } \neg x_i \text{ is in } C_j. \end{cases}$$

A recognizer $\Pi_{\mathcal{TFPPM}(m,n)}$ system is defined as follows:

$$\Pi_{\mathcal{TFPPM}(m,n)} = (O, P, \Sigma, \mu, \varnothing, w_1/z_1, \ldots, w_4/z_4, R, e, i_{in}, i_{out}),$$

where

- $O = \Sigma \cup \{a_i | 1 \le i \le n+1\}$
  $\cup \{t_{i,j}, f_{i,j}, b_i | 1 \le i \le n, 1 \le j \le m+1\}$
  $\cup \{r_j | 1 \le j \le m\}$
  $\cup \{s_j | 2 \le j \le m+1\}$
  $\cup \{c, \mathcal{YES}, \mathcal{NO}\}$;
- $P = \{g, g', p, p', q, s, s'\}$;
- $\Sigma = \{D_{i,j}, E_{i,j}, F_{i,j} | 1 \le i \le n, 1 \le j \le m\}$;
- $\mu = [[[\,]_3]_2[\,]_4]_1$;
- $w_1 = \{\mathcal{NO}\}, w_2 = \{a_1\}, w_3 = w_4 = \varnothing$;
- $z_1 = g, z_2 = p, z_3 = q, z_4 = s$;
- $i_{in} = 3$, $i_{out} = 0$ (the environment);
- $R$ is the following set of rules:

### 1) GENERATION PHASE

$$\begin{aligned}
R_{1,i} &\equiv [p|a_i]_2 \to [p|t_{i,1}]_2[p|f_{i,1}]_2, \quad i \in \{1, \ldots, n\}. \\
R_{2,i,j} &\equiv t_{i,j}[q|D_{i,j}]_3 \to t_{i,j+1}[q|r_j]_3, \\
&\quad t_{i,j}[q|E_{i,j}]_3 \to t_{i,j+1}[q|c]_3, \\
&\quad t_{i,j}[q|F_{i,j}]_3 \to t_{i,j+1}[q|c]_3, \\
&\quad i \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, m\} \\
R_{3,i,j} &\equiv f_{i,j}[q|E_{i,j}]_3 \to t_{i,j+1}[q|r_j]_3, \\
&\quad f_{i,j}[q|D_{i,j}]_3 \to t_{i,j+1}[q|c]_3, \\
&\quad f_{i,j}[q|F_{i,j}]_3 \to t_{i,j+1}[q|c]_3, \\
&\quad i \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, m\} \\
R_{4,i} &\equiv [p|t_{i,m+1}]_2 \to t_{i,m+1}[p'|\,]_2, i \in \{1, \ldots, n\}. \\
R_{5,i} &\equiv t_{i,m+1}[p_i|f_{i,m+1}]_2 \to b_i[p_i'|c]_2, \quad i \in \{1, \ldots, n\}. \\
R_{6,i} &\equiv b_i[s|\,]_4 \to [s'|b_i]_4, \quad i \in \{1, \ldots, n\}. \\
R_{7,i} &\equiv [s'|b_i]_4 \to [s'|a_{i+1}]_4[s'|a_{i+1}]_4, \quad i \in \{1, \ldots, n\}. \\
R_{8,i} &\equiv [s'|a_i]_4 \to a_i[s|\,]_4, \quad i \in \{2, \ldots, n+1\}. \\
R_{9,i} &\equiv a_{i+1}[p'|\,]_2 \to [p|a_{i+1}]_2, \quad i \in \{1, \ldots, n\}.
\end{aligned}$$

Initially, the system runs with applying rule $R_{1,1}$ and $R_{12}$ simultaneously. By applying division rule $R_{1,1}$, object $t_{1,1}$ (true of variable $x_1$) and $f_{1,1}$ (false of variable $x_1$) appear in the generated membranes. After rule $R_{1,1}$ is executed, by applying rule $R_{2,1,j}$ (resp., $R_{3,1,j}$ ), object $D_{1,j}$ (resp., $E_{1,j}$ ) evolves to object $r_j$ under the influence of object $t_{1,1}$ (resp., $f_{1,1}$). $\mathcal{TFPPM}$ systems apply rule $R_{2,1,j}$ (resp., $R_{3,1,j}$) with iterative process. By applying rule $R_{2,1,j}$ (resp., $R_{3,1,j}$), the second subscript of object $t_{1,j}$ (resp., $f_{1,j}$) in membrane 2 would add 1. Thus, when these rules finish, the subscript is $m+1$.

After rule $R_{2,1,j}$ is executed, object $t_{1,m+1}$ is generated, and rule $R_{4,1}$ is applied, thus object $t_{1,m+1}$ will appear outside of membrane 2. Simultaneously, the state of protein in that membrane can be changed from $p$ to $p'$. Next step, if object $f_{1,m+1}$ appears in the membrane, it indicates that all the rules in $R_{3,1,j}$ have been executed. At that time, object $t_{1,m+1}$ enters membrane 2 and evolves to object $c$ by using $R_{5,1}$. Simultaneously, object $f_{1,m+1}$ can be generated outside of membrane 2 and changed to object $b_1$. Next step, object $b_1$ enters membrane 4 by applying the next rule, changing the state of protein $s$ to $s'$. If object $b_1$ appears in the membrane 4, rule $R_{7,1}$ starts to be applied. Next step, object $a_2$ comes to the corresponding membrane, and the protein $s$ is generated by applying rule $R_{8,1}$. Finally, each copy of object $a_2$ enters each membrane with label 2 by applying rule $R_{9,1}$. With applying the rule, the computing process of $x_1$ completes.

The subsequent process works similarly with $x_1$. $\mathcal{TFPPM}$ systems continue to run for variables $x_2$. For the $i$-th iteration of variable $x_i$, rule $R_{5,i}$ start to apply until objects $t_{1,m+1}$ and $f_{1,m+1}$ appear in the corresponding membranes. Therefore, rule $R_{5,i}$ has a synchronization function.

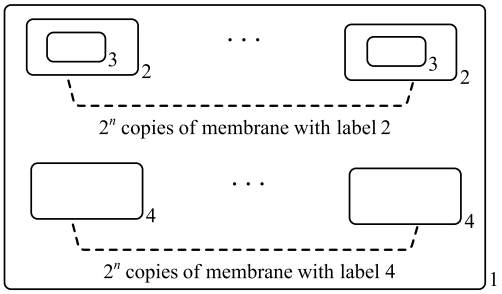



**FIGURE 2.** The membrane structure when generation phase halts.

It must be emphasized that $\mathcal{TFPPM}$ systems work in the time-free way. Furthermore, for each protein in the system, there exist only two types of working states. On the whole, after $2mn+6n$ RS-steps, the computing process of this phase completes, $2^n$ copies of membrane 2 and membrane 4 are generated in membrane 1 (see Figure 2).

#### 2) CHECKING PHASE

$$R_{10} \equiv\ a_{n+1}[q|r_1]_3 \rightarrow s_2[q|c]_3.$$
$$R_{11,j} \equiv\ s_j[q|r_j]_3 \rightarrow s_{j+1}[q|c]_3, j \in \{2,\ldots,m\}.$$

When object $a_{n+1}$ appears on an arbitrary membrane 2, it is obvious that the generation phase has finished. Because of maximal parallelism, object $a_{n+1}$ of each membrane 2 will appear at the same time. If $a_{n+1}$ in a membrane 2 and $r_1$ in a membrane 3, object $s_2$ can be generated in the membranes 2 with applying rule $R_{10}$.

If object $s_i$ $(2 \leq i \leq m)$ occurs, rule $R_{11,j}$ would be executed. When $s_m$ occurs, rule $R_{11,m}$ is executed. Therefore, $s_{m+1}$ may appear in the end.

#### 3) OUTPUT PHASE

$$R_{12} \equiv\ [g|\mathcal{NO}]_1 \rightarrow \mathcal{NO}[g'|\ ]_1.$$
$$R_{13} \equiv\ [p|s_{m+1}]_2 \rightarrow \mathcal{YES}[p'|\ ]_2.$$
$$R_{14} \equiv\ [g'|\mathcal{YES}]_1 \rightarrow \mathcal{YES}[g|\ ]_1.$$
$$R_{15} \equiv\ \mathcal{NO}[g|\ ]_1 \rightarrow [g'|\mathcal{NO}]_1.$$

Under the influence of the protein $g$ and object $\mathcal{NO}$, $R_{12}$ would be activated. At this moment, there exist two cases.

(i) affirmative answer: $s_{m+1}$ appears on an arbitrary membrane 2, and object $s_{m+1}$ is changed to $\mathcal{YES}$ and comes out from its own membrane by employing $R_{13}$. Next step, object $\mathcal{YES}$ come to the output region and the protein on the membrane 1 is changed to $g$. After the rules from $R_{12}$ to $R_{14}$ have been executed, under the influence of protein $g$, object $\mathcal{NO}$ comes to membrane 1. Therefore, it is a affirmative answer because $\mathcal{YES}$ exists in $i_{out}$ when the system reaches completion at the final configuration.

(ii) negative answer: In this case, the system cannot apply the rules from $R_{13}$ to $R_{15}$. Therefore, it is a affirmative answer because $\mathcal{NO}$ exists in $i_{out}$ when the system reaches completion at the final configuration.

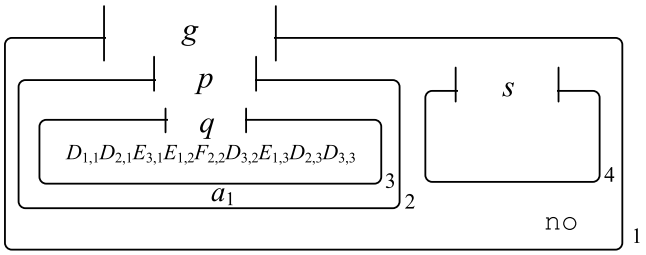



**FIGURE 3.** The initial configuration.

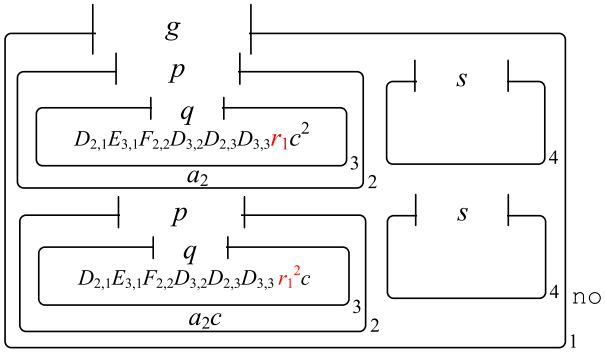



**FIGURE 4.** The configuration corresponding to the computation of $x_1$.

### B. SOME FORMAL DETAILS

The computing resources of $\Pi_{\mathcal{TFPPM}(m,n)}$ are listed as follows.

- size of the set $O$: $5mn+4n+2m+4 \in O(mn)$;
- size of the set $P$: $7 \in O(n)$;
- initial number of membranes: $4 \in O(1)$;
- initial number of objects: $2 \in O(1)$;
- initial number of proteins on membranes: $n+3 \in O(n)$;
- the total number of rules: $2mn+7n+m+4 \in O(mn)$;
- the maximal length of rules: $4 \in O(1)$.

## VI. AN INSTANCE OF $\Pi_{\mathcal{TFPPM}(M,N)}$

In this section, we use $\Pi_{\mathcal{TFPPM}(m,n)}$ to solve an instance, which is expressed as follows:

$$\gamma = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

The instance $\gamma$ is encoded by the multiset:

$$\Sigma = D_{1,1}D_{2,1}E_{3,1}E_{1,2}F_{2,2}D_{3,2}E_{1,3}D_{2,3}D_{3,3}$$

Figure 3 shows the initial structure of $\Pi_{\mathcal{TFPPM}(m,n)}$. At step 1, the system runs with applying rule $R_{1,1}$ and $R_{12}$ simultaneously, and membrane 2 is divided into two membranes by applying division rule $R_{1,1}$, so $t_{1,1}$ and $f_{1,1}$ can be generated in the new membrane respectively. Then, by applying rule $R_{2,1,j}$ (resp., $R_{3,1,j}$), $r_j$ or $e_1$ can be generated. The rule $R_{2,1,j}$ (resp., $R_{3,1,j}$) is used, thus $t_{1,4}$ (resp., $f_{1,4}$) will appear. Rule $R_{4,1}$ is activated when rule $R_{2,1,j}$ completes. Similarly, rule $R_{5,1}$ starts to be applied only when the application of $R_{2,1,j}$ and $R_{4,1}$ has finished. At that moment, $t_{1,4}$ and $f_{1,4}$ appear in the corresponding membrane respectively. Finally, object $b_1$ can be generated in membrane 1. When $b_1$ appears, rule $R_{7,1}$ is applied. Finally,

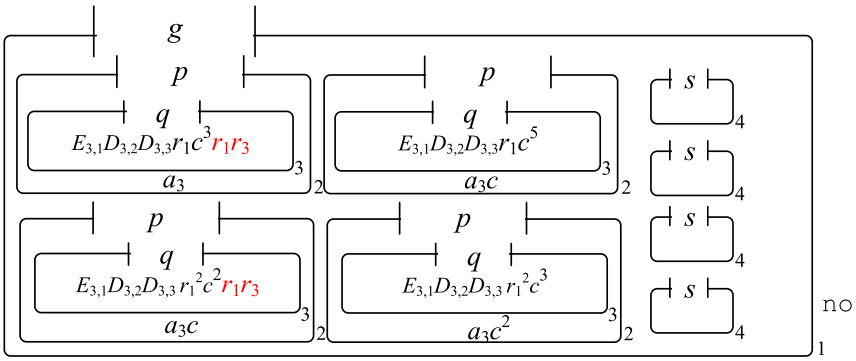



**FIGURE 5.** The configuration corresponding to the computation of $x_2$.

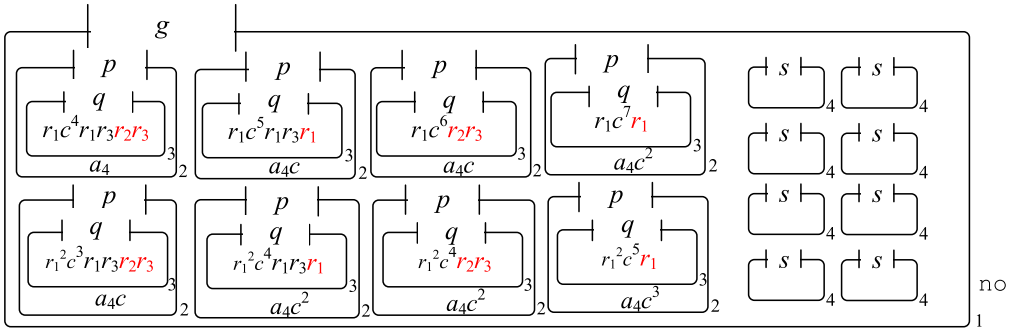



**FIGURE 6.** The configuration corresponding to the computation of $x_3$.

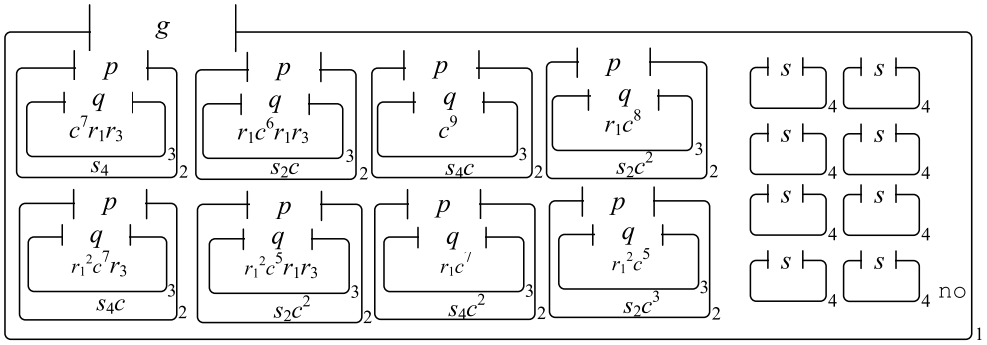



**FIGURE 7.** The configuration if checking phase finishes.

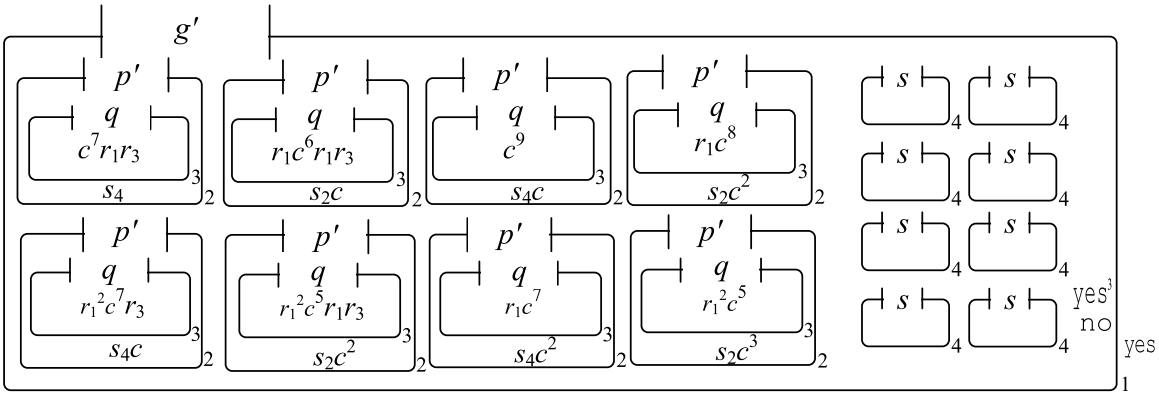



**FIGURE 8.** The final configuration.

when the application of $R_{8,1}$ and $R_{9,1}$ halts, object $a_2$ locates in membrane 2, and the computing process of $x_1$ halts (see Figure 4). Because time-freeness is the strategy for applying rules, we could not decide whether rule $R_{12}$ halts or not; here, we assume that the rule has been executed.

Once the computation of $x_1$ finishes, by using rule $R_{1,2}$, four membranes 2 are generated. $\Pi_{\mathcal{TFPPM}(m,n)}$ continues to assign values to variables $x_2$. similarly, Figure 5 (resp., Figure 6) is the configuration corresponding to the computing process of $x_2$ (resp., $x_3$).

If object $a_4$ appears on a membrane labeled 2, it is obvious that the generation phase has finished. When object $a_4$ and $r_1$ appear in membranes 2, object $s_2$ can be generated by applying rule $R_{10}$. Next, when $s_i$ occurs, the system starts to apply rule $R_{11,j}$. In the end, $s_4$ is generated (see Figure 7). When $\Pi_{\mathcal{TFPPM}(m,n)}$ reaches completion, the instance has satisfiable solution because $\mathcal{YES}$ exists in the output region (see Figure 8).

## VII. CONCLUSION

In this work, the variant $\mathcal{TFPPM}$ has been established, and we have obtained the result that only one membrane and a maximum rule length of 4 is Turing universal. Therefore, the model can be deployed in the time-free mode to actualize applications, thereby building robust computational system that would not be influenced by rule execution time. Additionally, we obtained a uniform solution of an **NP**-complete problem in polynomial time. It is shown that $\mathcal{TFPPM}$ has strong computational efficiency even when it runs with time-freeness. In [37], the $\mathcal{SAT}$ problem was solved by $\mathcal{PPM}$ systems with time-freeness; however, the protein on a membrane may has a variety of working states, which differs from our work; that is, our model has only two working states for each protein.

In Section IV, we have employed the rule types (c), (d) and (e). In order to optimize the result, readers can attempt to apply other rules to achieve Turing universality, e.g., types from (a) to (d), which have a shorter rule length.

In our work, when rules are applied, the time-free mode is adopted as the strategy. One can apply flat maximal parallelism [41], minimal parallelism [42], rule synchronization [43] and local synchronization [44] to apply rules.

## REFERENCES


[1] G. Paŭn, "Computing with membranes," *J. Comput. Syst. Sci.*, vol. 61, no. 1, pp. 108–143, Aug. 2000.

[2] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, Nov. 1994.

[3] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.

[4] R. Freund, G. Păun, and M. J. Pérez-Jiménez, "Tissue P systems with channel states," *Theor. Comput. Sci.*, vol. 330, no. 1, pp. 101–116, Jan. 2005.

[5] X. Song, L. Valencia-Cabrera, H. Peng, and J. Wang, "Spiking neural P systems with autapses," *Inf. Sci.*, vol. 570, pp. 383–402, Sep. 2021.

[6] T. Wu, Q. Lyu, and L. Pan, "Evolution-communication spiking neural P systems," *Int. J. Neural Syst.*, vol. 31, no. 2, Feb. 2021, Art. no. 2050064.

[7] T. Wu, L. Zhang, Q. Lyu, and Y. Jin, "Asynchronous spiking neural P systems with local synchronization of rules," *Inf. Sci.*, vol. 588, pp. 1–12, Apr. 2022.

[8] B. Song, X. Zeng, M. Jiang, and M. J. Pérez-Jiménez, "Monodirectional tissue P systems with promoters," *IEEE Trans. Cybern.*, vol. 51, no. 1, pp. 438–450, Jan. 2021.

[9] B. Song, X. Zeng, and A. Rodríguez-Patón, "Monodirectional tissue P systems with channel states," *Inf. Sci.*, vol. 546, pp. 206–219, Feb. 2021.

[10] Y. Luo, H. Tan, Y. Zhang, and Y. Jiang, "The computational power of timed P systems with active membranes using promoters," *Math. Struct. Comput. Sci.*, vol. 29, no. 5, pp. 663–680, May 2019.

[11] P. Guo, J. Zhu, H. Chen, and R. Yang, "A linear-time solution for all-SAT problem based on P system," *Chin. J. Electron.*, vol. 27, no. 2, pp. 367–373, Mar. 2018.

[12] P. Guo, C. Quan, and H. Chen, "MEAMVC: A membrane evolutionary algorithm for solving minimum vertex cover problem," *IEEE Access*, vol. 7, pp. 60774–60784, 2019.

[13] Y. Luo, Y. Zhao, and C. Chen, "Homeostasis tissue-like P systems," *IEEE Trans. Nanobiosci.*, vol. 20, no. 1, pp. 126–136, Jan. 2021.

[14] D. Díaz-Pernil, H. A. Christinal, and M. A. Gutiérrez-Naranjo, "Solving the 3-COL problem by using tissue P systems without environment and proteins on cells," *Inf. Sci.*, vols. 430–431, pp. 240–246, Mar. 2018.

[15] P. Guo, H. Z. Chen, and H. Zheng, "Arithmetic expression evaluations with membranes," *Chin. J. Electron.*, vol. 23, no. 1, pp. 55–60, 2014.

[16] X. Zeng, T. Song, X. Zhang, and L. Pan, "Performing four basic arithmetic operations with spiking neural P systems," *IEEE Trans. Nanobiosci.*, vol. 11, no. 4, pp. 366–374, Dec. 2012.

[17] P. Guo, J. Ji, H. Chen, and R. Liu, "Evaluating logical expressions by membrane systems," *Chin. J. Electron.*, vol. 23, no. 2, pp. 278–283, 2014.

[18] Y. Luo, P. Guo, Y. Jiang, and Y. Zhang, "Timed homeostasis tissue-like P systems with evolutional symport/antiport rules," *IEEE Access*, vol. 8, pp. 131414–131424, 2020.

[19] Z. Gazdag and G. Kolonits, "A new method to simulate restricted variants of polarizationless P systems with active membranes," *J. Membrane Comput.*, vol. 1, no. 4, pp. 251–261, Dec. 2019.

[20] H. Peng and J. Wang, "Coupled neural P systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1672–1682, Jun. 2019.

[21] L. Pan and B. Song, "P systems with rule production and removal," *Fundamenta Informaticae*, vol. 171, nos. 1–4, pp. 313–329, Oct. 2019.

[22] G. Singh and K. Deep, "Effectiveness of new multiple-PSO based membrane optimization algorithms on CEC 2014 benchmarks and Iris classification," *Natural Comput.*, vol. 16, no. 3, pp. 473–496, Sep. 2017.

[23] G. Zhang, H. Rong, F. Neri, and M. J. Pérez-Jiménez, "An optimization spiking neural P system for approximately solving combinatorial optimization problems," *Int. J. Neural Syst.*, vol. 24, no. 5, Aug. 2014, Art. no. 1440006.

[24] J. Wang, H. Peng, W. Yu, J. Ming, M. J. Pérez-Jiménez, C. Tao, and X. Huang, "Interval-valued fuzzy spiking neural P systems for fault diagnosis of power transmission networks," *Eng. Appl. Artif. Intell.*, vol. 82, pp. 102–109, Jun. 2019.

[25] J. Hu, H. Peng, J. Wang, and W. Yu, "kNN-P: A kNN classifier optimized by p systems," *Theor. Comput. Sci.*, vol. 817, pp. 55–65, May 2020.

[26] G. Zhang, M. J. Pérez-Jiménez, and M. Gheorghe, *Real-Life Applications With Membrane Computing*. Berlin, Germany: Springer, 2017.

[27] G. Zhang, M. J. Pérez-Jiménez, A. Riscos-Núñez, S. Verlan, S. Konur, T. Hinze, and M. Gheorghe, *Membrane Computing Models: Implementations*. Berlin, Germany: Springer, 2021.

[28] B. Song, K. Li, D. Orellana-Martín, M. J. Pérez-Jiménez, and I. Pérez-Hurtado, "A survey of nature-inspired computing: Membrane computing," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 1–31, 2021.

[29] A. Păun and B. Păun, "P systems with proteins on membranes," *Int. J. Found. Comput. Sci.*, vol. 72, no. 4, pp. 467–483, 2006.

[30] A. Păun, M. Paun, A. Rodríguez-Patón, and Manuela Sidoroff, "P systems with proteins on membranes: A survey," *Int. J. Found. Comput. Sci.*, vol. 22, no. 1, pp. 39–53, 2011.

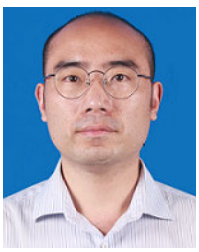
[31] A. Păun and B. Popa, "P systems with proteins on membranes and membrane division," in *Developments in Language Theory* (Lecture Notes in Computer Science). Berlin, Germany: Springer-Verlag, 2006, pp. 292–303.

[32] P. Sosík, A. Păun, and A. Rodríguez-Patón, "P systems with proteins on membranes characterize PSPACE," *Theor. Comput. Sci.*, vol. 488, pp. 78–95, Jun. 2013.

[33] C. Hu, Y. Li, and B. Song, "P systems with proteins on active membranes," *Mathematics*, vol. 10, no. 21, pp. 1–10, 2022.
[34] M. Cavaliere and D. Sburlan, "Time–independent P systems," in *Membrane Computing* (Lecture Notes in Computer Science). Berlin, Germany: Springer-Verlag, 2005, pp. 239–258.
[35] Y. Luo, Z. Xiong, and G. Zhang, "Time-free solution to SAT problem by tissue P systems," *Math. Problems Eng.*, vol. 2017, pp. 1–8, Jan. 2017.
[36] Y. Zhao, X. Liu, M. Sun, J. Qu, and Y. Zheng, "Time-free cell-P systems with multiple promoters/inhibitors," *Theor. Comput. Sci.*, vol. 843, pp. 73–83, Dec. 2020.
[37] B. Song, M. J. Pérez-Jiménez, and L. Pan, "An efficient time-free solution to SAT problem by P systems with proteins on membranes," *J. Comput. Syst. Sci.*, vol. 82, no. 6, pp. 1090–1099, Sep. 2016.
[38] S. Krishna, "On the computational power of flip-flop proteins on membranes," in *Proc. Conf. Comput. Eur., Comput. Log. Real World*. Bombay, India: Springer-Verlag, 2007, pp. 695–704.
[39] A. Păun and A. Rodríguez-Patón, "On flip-flop membrane systems with proteins," in *Proc. Int. Conf. Membrane Comput.* Thessaloniki, Greece: Springer-Verlag, 2007, pp. 414–427.
[40] M. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1967.
[41] L. Pan, G. Păun, and B. Song, "Flat maximal parallelism in P systems with promoters," *Theor. Comput. Sci.*, vol. 623, pp. 83–91, Apr. 2016.
[42] G. Ciobanu, L. Pan, G. Păun, and M. J. Pérez-Jiménez, "P systems with minimal parallelism," *Theor. Comput. Sci.*, vol. 378, no. 1, pp. 117–130, Jun. 2007.
[43] B. Song and L. Pan, "Rule synchronization for tissue P systems," *Inf. Comput.*, vol. 281, Dec. 2021, Art. no. 104685.
[44] L. Pan, A. Alhazov, H. Su, and B. Song, "Local synchronization on asynchronous tissue P systems with symport/antiport rules," *IEEE Trans. Nanobiosci.*, vol. 19, no. 2, pp. 315–320, Apr. 2020.

**XIAOMING WAN** was born in Chongqing, China, in 1975. He received the master's degree in computer application from Chongqing University, China, in 2009. He is currently with the College of Electronics and IoT Engineering, Chongqing Industry Polytechnic College, Chongqing. His research interests include biological computing and intelligent information processing.

**YI LIU** was born in Chongqing, China, in 1977. He is currently a Professor with the College of Rail Transit and Aviation Services, Chongqing Industry Polytechnic College, Chongqing. His research interests include big data, network security, and intelligent information processing.



**YUEGUO LUO** was born in Gulin, Sichuan, China, in 1979. He received the Ph.D. degree from Chongqing University, China, in 2017. Since 2004, he has been a Teacher with Yangtze Normal University, Fuling, Chongqing. He is currently an Associate Professor. His research interests include membrane computing and computational intelligence.