International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 12, Number 2, pages 238–252, July 2022

An asynchronous P system with a DPLL algorithm for solving SAT

Takuya Noguchi Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan

Received: February 5, 2022
Accepted: March 22, 2022
Communicated by Susumu Matsumae

Abstract

Membrane computing, which is also known as P system, is a computational model inspired by the activity of living cells. Several efficient P systems, which work in a polynomial number of steps, have been proposed for solving computationally hard problems. However, most of the proposed algorithms use an exponential number of membranes, and reduction of the number of membranes must be considered in order to make the P system a more realistic model.

In the present paper, we propose an asynchronous P system with a Davis-Putnam-Logemann-Loveland (DPLL) algorithm, which is a set of rules for solving a satisfiability problem (SAT) efficiently, in an attempt to reduce the number of membranes. The proposed P system solves SAT with n variables and m clauses in $O(mn^2)$ parallel steps or $O(mn^2 2^n)$ sequential steps.

We evaluate the number of membranes used in the proposed P system by comparison with the number of membranes used in known P systems. The experimental result demonstrates the validity and efficiency of the proposed P system.

Keywords: membrane computing, satisfiability problem, DPLL algorithm

1 Introduction

Membrane computing, which was introduced in [8] as P system, is a computational model inspired by the activity of living cells. The definition of P system is based on a feature of a membrane and an object, which denote a computing cell and data storage, respectively. In P system, each object evolves according to evolution rules, which are associated with the membrane.

Since an exponential number of membranes can be created in a polynomial number of steps using the division rule, which is one of evolution rules in a P system, a computationally hard problem can be solved in a polynomial number of steps. Using this feature, a number of P systems have been proposed for solving computationally hard problems [4, 5, 7, 9, 12, 14].

In addition, asynchronous parallelism, which assumes asynchronous application of evolution rules, has been considered for P systems. Asynchronous parallelism means that all objects may react to rules with different speeds on P system. Asynchronous parallelism makes P system a more realistic computational model because living cells work independently according to the environment. Using asynchronous parallelism, a number of asynchronous P systems have been proposed for computationally hard problems [2, 10, 11].

However, computationally hard problems have been solved in a polynomial number of steps using an exponential number of membranes. The number of membranes indicates the number of living cells, and a reduction of the number of membranes must be considered in the case where the P system is implemented using living cells because living cells cannot be created exponentially.

Recently, a number of P systems [3, 6, 13] have been proposed for reducing the number of membranes. For example, P system for the satisfiability problem (SAT) [3] with branch and bound, which is a well-known optimization technique, has been proposed. In the proposed P system, the satisfiability is checked for partial assignment, and the partial assignment is bounded if all clauses are satisfied or if one of the clauses cannot be satisfied. The experimental results for the proposed P system show that the number of membranes used is at most 75 percent less than the number of membranes used in previous P system for SAT.

In the present paper, we propose an asynchronous P system for solving SAT with the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. The DPLL algorithm is a well-known search algorithm for SAT and preferentially assigns a variable in the partial assignment using three rules: a one-literal rule, a pure literal rule, and a splitting rule. Using these three rules repeatedly and intensively, the number of membranes that contain partial assignment can be decreased.

We propose an asynchronous P system for solving SAT with n variables and m clauses using the DPLL algorithm. The theoretical complexities of proposed P system is $O(mn^2 2^n)$ sequential steps or $O(mn^2)$ parallel steps using $O(m^2 n^2)$ kinds of objects.

We evaluate the number of membranes in the proposed P system by comparing the number of membranes in known P systems. The experimental results show that the number of membranes used is nearly constant for small inputs and is smaller than the number of membranes used in previous P systems.

The remainder of the paper is organized as follows. In Section 2, we describe the computational model for the membrane computing and an outline of the DPLL algorithm. In Section 3, we propose P system with the DPLL algorithm for SAT and consider a complexity of P system. In Section 4, we show experimental results for the previous P systems and proposed P systems. Finally, Section 5 concludes the paper.

2 Preliminaries

In the present paper, we propose an asynchronous P system with the DPLL algorithm. We briefly explain the definition of P system and an outline of the DPLL algorithm in this section.

2.1 Computational model for membrane computing

A P system consists mainly of membranes and objects. A membrane is a computing cell in the P system and may contain objects and other membranes. Each membrane is labeled with a distinct integer. An object denotes a memory cell that stores data in the P system. According to the evolution rules for the corresponding membrane, objects may evolve into other objects or pass through membranes. Objects may also divide or dissolve the membranes in which the objects are stored. We assume that each object is a finite string over a given set of alphabetic characters.

As an example of membranes and objects, the following expression denotes a membrane structure that consists of two membranes and three objects.

$$[\ [\ \alpha\ ]_2\ [\ \beta\ \gamma\ ]_3\ ]_1$$

In this example, the membrane labeled 1 contains two membranes, labeled 2 and 3, and the membranes labeled 2 and 3 contain sets of objects $\{\alpha\}$ and $\{\beta, \gamma\}$, respectively.

Computation of P systems is governed by a number of evolution rules. Each evolution rule is a rewriting rule for membranes and objects. According to the applicable evolution rules, objects and membranes are transformed in parallel in every step of computation. The system stops computation if there is no applicable evolution rule for the objects.

A number of types of evolution rules are assumed in membrane computing. In the present paper, we assume the following five rules as in [4]:

(1) Object evolution rule:

$$[\ \alpha\ ]_h \rightarrow [\ \beta\ ]_h$$

Object α is transformed into object β.

(2) Send-in communication rule:

$$\alpha\ [\ ]_h \rightarrow [\ \beta\ ]_h$$

Object α is moved into inner membrane h and is transformed into object β.

(3) Send-out communication rule:

$$[\ \alpha\ ]_h \rightarrow [\ ]_h\ \beta$$

Object α is sent out from membrane h and is transformed into object β.

(4) Dissolution rule:

$$[\ \alpha\ ]_h \rightarrow \beta$$

The membrane that contains object α is dissolved, and object α is transformed into object β. (Note that the outermost membrane cannot be dissolved.)

(5) Division rule:

$$[\ \alpha\ ]_h \rightarrow [\ \beta\ ]_h[\ \gamma\ ]_h$$

The membrane that contains object α is divided into two membranes with the same label, and object α is transformed into other objects, β and γ, in each of the divided membranes.

The P system consists of the following six components:

O**:** the set of objects used in the system,

μ**:** the structure of the membrane,

ω_i**:** the set of objects initially contained in the membrane labeled i,

R_i**:** the set of evolution rules for the membrane labeled i,

i_{in}**:** the label of the input membrane, and

i_{out}**:** the label of the output membrane

Using the above components, a P system Π with m membranes is defined as follows:

$$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m, R_1, R_2, \cdots, R_m, i_{in}, i_{out})$$

The complexity of the P system is defined as follows. We assume that each of the evolution rules can be applied in one step in the computational model, and the complexity of the P system is the number of steps executed.

In the present paper, we consider asynchronous parallelism [10] in the P system. Under the assumption of asynchronous parallelism, any number of applicable evolution rules is applied in parallel. In other words, all objects, for which there are applicable evolution rules, can be transformed in parallel, or only one of the applicable evolution rules is applied in each step of computation. We refer to the number of steps in the former and latter cases as the number of parallel and sequential steps, respectively. The number of parallel steps is the complexity of the P system in the best case, and the number of sequential steps is the complexity in the worst case.

2.2 DPLL algorithm for CNF-SAT

In this subsection, we present an outline of the DPLL algorithm [1], which is a known search algorithm for solving CNF-SAT. CNF-SAT is a well-known computationally hard problem that determines if there exists a truth assignment for a given Boolean formula in conjunctive normal form. The DPLL algorithm for SAT consists of three rules: a one-literal rule, a pure literal rule, and a splitting rule. The rules are based on eliminating useless variable assignments.

(a) One-literal rule

The one-literal rule focuses on a clause containing only one literal. The clause can only be satisfied by assignments such that the literal is set as true. The following is an example of an input formula for the one-literal rule:

$$L = (X_1 \vee \neg X_2) \wedge (X_1 \vee X_2) \wedge (\neg X_1)$$

In the above case, the third clause contains a single literal. Using the one-literal rule for the third clause, X_1 is set to 0.

(b) Pure literal rule

The pure literal rule focuses on a pure literal, which is a literal with only one polarity in the input formula. A pure literal can be assigned as true or false according to polarity of the literal and can be deleted from clauses in which the literal is contained. The following is an example of an input formula for the pure literal rule:

$$L = (X_1 \vee \neg X_2) \wedge (X_1 \vee X_2) \wedge (X_1 \vee X_3)$$

In the above case, X_1 is a positive literal in all clauses, and the literal is a pure literal. Using the pure literal rule, X_1 is set to 1.

(c) Splitting rule

A non-pure literal x, which is positive or negative in terms of clauses, can be assigned as $x = 0$ or $x = 1$. The splitting rule is applied for the literal, and the input formula is divided into two formulas according to the two assignments. For example, the following is an input formula that is divided into two formulas by applying the splitting rule to X_3:

$$\begin{aligned} L &= (X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3) \\ &= \begin{cases} X_1 \vee \neg X_2 & (X_3 = 0) \\ \neg X_1 \vee X_2 & (X_3 = 1) \end{cases} \end{aligned}$$

Using the above three rules, we can solve SAT effectively using the following procedure.

1. If a given formula consists of a single variable, then the satisfiability of the formula is determined in a constant time.
2. If the one-literal rule or the pure literal rule is applicable to a given formula, then the rules are applied for the formula. If these two rules cannot be applied, then the splitting rule is applied for the formula.
3. After applying the splitting rule, the input formula is divided into two formulas. Then, the procedure is executed recursively for the two formulas.

3 An asynchronous P system with the DPLL algorithm for SAT

In this section, we explain the proposed asynchronous P system for solving SAT. We first show an encoding for the input and the output for the P system and then present an outline and details of the P system. We next show an example of execution of the proposed P system, and we finally discuss the complexity of the proposed P system.

3.1 Input and output for proposed P system

We assume that an input formula for SAT is given in conjunctive normal form (CNF) with n Boolean variables and m clauses. We also assume that an output for SAT is one of two values, "TRUE" or "FALSE". The output is "TRUE" if there exists a truth assignment for satisfying the formula; otherwise, the output is "FALSE".

The following is an example of an input formula with three variables and three clauses. Since a truth assignment, $X_1 = 0$, $X_2 = 1$ and $X_3 = 0$, satisfies the input formula, an output of SAT for this instance is "TRUE".

$$L = (X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_3) \wedge (\neg X_1)$$

The above input is given by the following set of objects O_L in the P system:

$$O_L = \{\langle X_{i,j}, V\rangle \mid 1 \leq i \leq n, 1 \leq j \leq m, V \in \{0, 1, N\}\}$$

Each object $\langle X_{i,j}, V\rangle$ denotes a Boolean value for variable X_i in the j-th clause. When X_i occurs in j-th clause, $V = 1$ and when $\neg X_i$ occurs in j-th clause, $V = 0$. In addition, V is set to N if neither X_i nor $\neg X_i$ is in the j-th clause. For example, the following set of objects denotes the above input formula:

$$\begin{aligned} O_L =\{&\langle X_{1,1}, 1\rangle, \langle X_{1,2}, 0\rangle, \langle X_{1,3}, 0\rangle, \\ &\langle X_{2,1}, 1\rangle, \langle X_{2,2}, N\rangle, \langle X_{2,3}, N\rangle, \\ &\langle X_{3,1}, N\rangle, \langle X_{3,2}, 0\rangle, \langle X_{3,3}, N\rangle\} \end{aligned}$$

We assume that the input set O_L is given from the outside region into the outer membrane. The output of the P system is one of two objects, $\langle TRUE\rangle$ or $\langle FALSE\rangle$. Object $\langle TRUE\rangle$ is sent out from the outer membrane to the outside region if the input formula is satisfiable; otherwise, object $\langle FALSE\rangle$ is sent out to the outside region.

3.2 Outline of P system with DPLL algorithm

The proposed P system consists of two membranes $[\ [\ ]_2\ ]_1$, i.e., an inner membrane labeled 2 is contained in an outer membrane labeled 1. The P system consists of the following six steps:

Step 1: Move a set of input objects into an inner membrane.

Step 2: In each inner membrane, choose a variable such that the pure literal rule is applicable to the variable. If the variable exists, then assign a Boolean value for the variable according to the pure literal rule.

Step 3: If no variable is chosen in Step 2, choose a variable such that the one-literal rule is applicable to the variable. If the variable exists, then assign a Boolean value for the variable according to the one-literal rule.

Step 4: If no variable is chosen in Steps 2 and 3, then divide the inner membrane into two membranes and assign a value for a variable in each divided inner membrane according to the splitting rule.

Step 5: Check the satisfiability in each inner membrane and send an object indicating "TRUE" to the outer membrane if all clauses are satisfied. On the other hand, send an object indicating "FALSE" to the outer membrane if one of the clauses cannot be satisfied. Otherwise, the above procedure is repeated from Step 2.

Step 6: Send one of the final objects, $\langle TRUE\rangle$ and $\langle FALSE\rangle$, from the outer membrane if objects indicating "TRUE" or "FALSE" are sent out from the inner membrane.

3.3 Details of proposed P system

We now explain the details of each step of the P system with the DPLL algorithm. In the following description, $R_{i,j}$ denotes a set of evolution rules applied to membrane i in Step j. A set of objects O_L, which denotes an input formula, is given to the outer membrane.

Step 1

In Step 1, a set of input objects is moved into an inner membrane. The step is executed using the following evolution rules:

(Evolution rules for the outer membranes)

$$\begin{aligned}
R_{1,1} \;=\; & \{\langle X_{1,1}, V\rangle[]_2 \to [\langle M_{2,1}, 1\rangle\langle X_{1,1}, V\rangle\langle A_1, N\rangle]_2 \mid V \in \{0,1\}\} \\
& \cup\{\langle X_{1,1}, N\rangle[]_2 \to [\langle M_{2,1}, 0\rangle\langle X_{1,1}, N\rangle\langle A_1, N\rangle]_2\} \\
& \cup\{\langle M_{i,1}, p\rangle\langle X_{i,1}, V\rangle[]_2 \to [\langle M_{i+1,1}, p+1\rangle\langle X_{i,1}, V\rangle\langle A_i, N\rangle]_2 \\
& \quad \mid 2 \le i \le n, 0 \le p \le n-1, V \in \{0,1\}\} \\
& \cup\{\langle M_{i,1}, p\rangle\langle X_{i,1}, N\rangle[]_2 \to [\langle M_{i+1,1}, p\rangle\langle X_{i,1}, N\rangle\langle A_i, N\rangle]_2\} \\
& \quad \mid 2 \le i \le n, 0 \le p \le n-1\} \\
& \cup\{\langle M_{i,j}, p\rangle\langle X_{i,j}, V\rangle[]_2 \to [\langle M_{i+1,j}, p+1\rangle\langle X_{i,j}, V\rangle]_2 \\
& \quad \mid 1 \le i \le n, 2 \le j \le m, 0 \le p \le n-1, V \in \{0,1\}\} \\
& \cup\{\langle M_{i,j}, p\rangle\langle X_{i,j}, N\rangle[]_2 \to [\langle M_{i+1,j}, p\rangle\langle X_{i,j}, N\rangle]_2 \\
& \quad \mid 1 \le i \le n, 2 \le j \le m, 0 \le p \le n-1\}
\end{aligned}$$

(Evolution rules for the inner membranes)

$$\begin{aligned}
R_{2,1} \;=\; & \{[\langle M_{i,j}, p\rangle]_2 \to []_2\langle M_{i,j}, p\rangle \mid 1 \le i \le n, 1 \le j \le m, 0 \le p \le n\} \\
& \cup\{\langle M_{n+1,j}, p\rangle \to \langle M_{1,j+1}, 0\rangle\langle F_j, p, 0\rangle \mid 1 \le j \le m, 0 \le p \le n\} \\
& \cup\{\langle M_{1,m+1}, 0\rangle \to \langle P_{1,1}\rangle\langle PUR, N\rangle\langle LIT, n\rangle\langle MEM, m\rangle\}
\end{aligned}$$

The computation using the above evolution rules is executed as follows. All input objects in the outer membrane are moved into the inner membrane using the object $\langle M_{i,j}, p\rangle$. When objects of 1st clause are moved, the object $\langle A_i, N\rangle$ is created. The object indicates assignment of i-th literal. In the movement, the number of unassigned literals in each clause is counted using a value p in the object $\langle M_{i,j}, p\rangle$. At the end of movement for the j-th clause, objects $\langle M_{1,j+1}, 0\rangle$ and $\langle F_j, p, 0\rangle$ are created, and the objects trigger movement for the $(j+1)$-st clause. The p and 0 in object $\langle F_j, p, 0\rangle$ indicate that p literals are in the j-th clause and the clause is not satisfied, respectively.

At the end of Step 1, object $\langle M_{1,m+1}, 0\rangle$ is created after all input objects are moved into the inner membrane. As the final task of Step 1, objects $\langle P_{1,1}\rangle$, $\langle PUR, N\rangle$, $\langle LIT, n\rangle$, and $\langle MEM, m\rangle$ are created. Then, objects $\langle P_{1,1}\rangle$ and $\langle PUR, N\rangle$ trigger Step 2. In addition, $\langle LIT, n\rangle$ and $\langle MEM, m\rangle$ are objects that denote the number of unassigned literals and the number of unsatisfied clauses, respectively.

Step 2

In Step 2, a variable to which pure literal rules can be applied are selected and a Boolean value is assigned to the variable according to the pure literal rule if the variable exists. The step is executed using the following evolution rules:

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,2,1} \;=\; & \{\langle P_{i,j}\rangle\langle X_{i,j},V\rangle\langle PUR,N\rangle\langle F_j,p,0\rangle \rightarrow \langle P_{i,j+1}\rangle\langle X_{i,j},V\rangle\langle PUR,V\rangle\langle F_j,p,0\rangle \\ & \quad \mid 1 \le i \le n, 1 \le p \le n, 1 \le j \le m, V \in \{0,1\}\} \\ & \cup\{\langle P_{i,j}\rangle\langle X_{i,j},V\rangle\langle PUR,V\rangle \rightarrow \langle P_{i,j+1}\rangle\langle X_{i,j},V\rangle\langle PUR,V\rangle \\ & \quad \mid 1 \le j \le m, 1 \le i \le n, V \in \{0,1\}\} \\ & \cup\{\langle P_{i,j}\rangle\langle X_{i,j},V\rangle\langle PUR,V'\rangle\langle F_j,p,0\rangle \rightarrow \langle P_{i+1,j}\rangle\langle X_{i,j},V\rangle\langle PUR,N\rangle\langle F_j,p,0\rangle \\ & \quad \mid 1 \le i \le n, 1 \le p \le n, 1 \le j \le m, V,V' \in \{0,1\}, V \neq V'\} \\ & \cup\{\langle P_{i,j}\rangle\langle X_{i,j},N\rangle \rightarrow \langle P_{i,j+1}\rangle\langle X_{i,j},N\rangle \mid 1 \le i \le n, 1 \le j \le m\} \\ & \cup\{\langle P_{i,j}\rangle\langle F_j,0,1\rangle \rightarrow \langle P_{i,j+1}\rangle\langle F_j,0,1\rangle \mid 1 \le i \le n, 1 \le j \le m\} \end{aligned}$$

$$\begin{aligned} R_{2,2,2} \;=\; & \{\langle P_{i,m+1}\rangle\langle PUR,V\rangle\langle A_i,N\rangle\langle LIT,k\rangle \rightarrow \langle A_i,V\rangle\langle LIT,k-1\rangle\langle C_{i,1},i\rangle\langle FALSE,2^{k-1}\rangle \\ & \quad \mid 1 \le i \le n, 1 \le k \le n, 1 \le j \le m, V \in \{0,1\}\} \\ & \cup\{\langle P_{i,m+1}\rangle\langle PUR,N\rangle \rightarrow \langle P_{i+1,1}\rangle\langle PUR,N\rangle \mid 1 \le i \le n\} \\ & \cup\{\langle P_{n+1,1}\rangle\langle PUR,N\rangle \rightarrow \langle N_1\rangle\} \end{aligned}$$

The computation using the above evolution rules is executed as follows. First, the value of the first literal in the first clause is stored in object $\langle PUR,N\rangle$ by object $\langle P_{1,1}\rangle$. Then, the value of the first literal in each clause is checked, and the check is moved to the next variable if value V of $\langle PUR,V\rangle$ is not equal to value V of $\langle X_{i,j},V\rangle$.

In the case where all clauses are checked and all values are equal to $\langle PUR,V\rangle$, the pure literal rule is applicable to the literal. Then, object $\langle P_{i,m+1}\rangle$ is created, and value V in object $\langle PUR,V\rangle$ is set to V in $\langle A_i,V\rangle$. This procedure means that a value of the i-th variable is set to V. In addition, object $\langle C_{i,1},i\rangle$, which triggers Step 5, is created.

In the case the pure literal rule cannot be applied to any variable, object $\langle N_1\rangle$, which triggers Step 3, is created.

Step 3

In Step 3, a variable is chosen such that the one-literal rule is applicable to the variable, and a Boolean value is assigned to the variable according to the one-literal rule if the variable exists. Step 3 is executed using the following evolution rules:

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,3,1} \;=\; & \{\langle N_j\rangle\langle F_j,p,V\rangle \rightarrow \langle N_{j+1}\rangle\langle F_j,p,V\rangle \mid 2 \le p \le n, 1 \le j \le m, V \in \{0,1\}\} \\ & \cup\{\langle N_j\rangle\langle F_j,V,1\rangle \rightarrow \langle N_{j+1}\rangle\langle F_j,V,1\rangle \mid 1 \le j \le m, V \in \{0,1\}\} \\ & \cup\{\langle N_j\rangle\langle F_j,1,0\rangle \rightarrow \langle K_{1,j}\rangle\langle F_j,1,0\rangle \mid 1 \le j \le m\} \cup \{\langle N_{m+1}\rangle \rightarrow \langle S_1\rangle\} \end{aligned}$$

$$\begin{aligned} R_{2,3,2} \;=\; & \{\langle K_{i,j}\rangle\langle X_{i,j},V\rangle\langle A_i,N\rangle\langle LIT,k\rangle \\ & \rightarrow \langle A_i,V\rangle\langle X_{i,j},V\rangle\langle LIT,k-1\rangle\langle C_{i,1},i\rangle\langle FALSE,2^{k-1}\rangle \\ & \quad \mid 1 \le i \le n, 1 \le k \le n, 1 \le j \le m, V \in \{0,1\}\} \\ & \cup\{\langle K_{i,j}\rangle\langle X_{i,j},V\rangle\langle A_i,V'\rangle\langle LIT,k\rangle \rightarrow \langle FALSE,2^k\rangle \\ & \quad \mid 1 \le i \le n, 1 \le k \le n, 1 \le j \le m, V,V' \in \{0,1\}, V \neq V'\} \\ & \cup\{\langle K_{i,j}\rangle\langle X_{i,j},N\rangle \rightarrow \langle K_{i+1,j}\rangle\langle X_{i,j},N\rangle\} \end{aligned}$$

The computation using the above evolution rules is executed as follows. First, the clause which consists of only one literal are checked by using value p in $\langle F_j,p,V\rangle$. In case that the value $p=1$, j-th clause consists of one literal. After finding the clause, object $\langle K_{1,j}\rangle$ is created for assigning the value, and the value of the literal is set to $\langle A_i,V\rangle$. In addition, object $\langle C_{i,1},i\rangle$, which triggers Step 5, is created.

In the case where the one-literal rule cannot be applied for any clause, object $\langle S_1 \rangle$, which triggers Step 4, is created.

Step 4

Step 4 is executed if no variable is chosen in Steps 2 and 3. In Step 4, the inner membrane is divided into two membranes, and the value of a variable is assigned to each divided inner membrane according to the splitting rule. Step 4 is executed using the following evolution rules:

(Evolution rules for the inner membrane)

$$\begin{aligned}
R_{2,4} \quad = \quad & \{[\langle S_i \rangle \langle A_i, N \rangle \langle LIT, k \rangle]_2 \rightarrow [\langle A_i, 0 \rangle \langle LIT, k-1 \rangle \langle C_{i,1}, i \rangle]_2 [\langle A_i, 1 \rangle \langle LIT, k-1 \rangle \langle C_{i,1}, i \rangle]_2 \\
& \quad \mid 1 \leq i \leq n, 1 \leq k \leq n\} \\
& \cup \{\langle S_i \rangle \langle A_i, V \rangle \rightarrow \langle S_{i+1} \rangle \langle A_i, V \rangle \mid 1 \leq i \leq n, V \in \{0,1\}\}
\end{aligned}$$

The computation using the above evolution rules is executed as follows. Object $\langle S_i \rangle$ triggers membrane division according to the splitting rule. In the division, object $\langle A_i, 0 \rangle$ is created in one membrane, and object $\langle A_i, 1 \rangle$ is created in another membrane. Simultaneously, object $\langle C_{i,1}, i \rangle$, which triggers Step 5, is created in both of the membranes.

Step 5

In Step 5, the satisfiability is checked in each inner membrane, and an object indicating "TRUE" is sent to the outer membrane if all clauses are satisfied. On the other hand, an object indicating "FALSE" is sent to the outer membrane if one of the clauses cannot be satisfied. Otherwise, the above procedure is repeated from Step 2.

Step 5 is executed using the following evolution rules:

(Evolution rules for the inner membrane)

$$\begin{aligned}
R_{2,5,1} \quad = \quad & \{\langle C_{i,j}, i \rangle \langle X_{i,j}, V \rangle \langle A_i, V' \rangle \langle F_j, p, 0 \rangle \rightarrow \langle C_{i,j+1}, i \rangle \langle X_{i,j}, N \rangle \langle A_i, V' \rangle \langle F_j, p-1, 0 \rangle \\
& \quad \mid 1 \leq i \leq n, 2 \leq p \leq n, 1 \leq j \leq m, V, V' \in \{0,1\}, V \neq V'\} \\
& \cup \{\langle C_{i,j}, i \rangle \langle X_{i,j}, N \rangle \langle A_i, V \rangle \langle F_j, p, V' \rangle \rightarrow \langle C_{i,j+1}, i \rangle \langle X_{i,j}, N \rangle \langle A_i, V \rangle \langle F_j, p, V' \rangle \\
& \quad \mid 1 \leq i \leq n, 1 \leq p \leq n, 1 \leq j \leq m, V, V' \in \{0,1\}\} \\
& \cup \{\langle C_{i,j}, i \rangle \langle X_{i,j}, V \rangle \langle A_i, V \rangle \langle F_j, p, 0 \rangle \langle MEM, q \rangle \\
& \quad \rightarrow \langle C_{i,j+1}, i \rangle \langle X_{i,j}, N \rangle \langle A_i, V \rangle \langle F_j, 0, 1 \rangle \langle MEM, q-1 \rangle \\
& \quad \mid 1 \leq i \leq n, 1 \leq p \leq n, 1 \leq j \leq m, 1 \leq q \leq m, V \in \{0,1\}\} \\
& \cup \{\langle C_{i,j}, i \rangle \langle X_{i,j}, V \rangle \langle A_i, V' \rangle \langle F_j, p, 1 \rangle \rightarrow \langle C_{i,j+1}, i \rangle \langle X_{i,j}, V \rangle \langle A_i, V' \rangle \langle F_j, 0, 1 \rangle \\
& \quad \mid 1 \leq i \leq n, 2 \leq p \leq n, 1 \leq j \leq m, V \in \{0,1\}, V' \in \{0,1,N\}\} \\
& \cup \{\langle C_{i,j}, i \rangle \langle X_{i,j}, V \rangle \langle A_i, V' \rangle \langle F_j, 1, V'' \rangle \langle LIT, k \rangle \rightarrow \langle FALSE, 2^k \rangle \\
& \quad \mid 1 \leq i \leq n, 1 \leq j \leq m, V, V', V'' \in \{0,1\}\} \\
& \cup \{\langle C_{i,j}, i \rangle \langle F_j, 0, 1 \rangle \rightarrow \langle C_{i,j+1}, i \rangle \langle F_j, 0, 1 \rangle \mid 1 \leq i \leq n, 1 \leq j \leq m\}
\end{aligned}$$

$$\begin{aligned}
R_{2,5,2} \quad = \quad & \{\langle C_{i,m+1}, i \rangle \rightarrow \langle CHECK \rangle \mid 1 \leq i \leq n\} \\
& \cup \{\langle MEM, q \rangle \langle CHECK \rangle \rightarrow \langle MEM, q \rangle \langle P_{1,1} \rangle \langle PUR, N \rangle \mid 1 \leq q \leq m\} \\
& \cup \{\langle MEM, 0 \rangle \langle CHECK \rangle \rightarrow \langle TRUE \rangle\} \\
& \cup \{[\langle FALSE, 2^k \rangle]_2 \rightarrow []_2 \langle FALSE, 2^k \rangle \mid 0 \leq k \leq n-1\}
\end{aligned}$$

The computation using the above evolution rules is executed as follows. Each clause is checked for satisfiability by checking object $\langle C_{i,j}, i \rangle$. In the case where the clause is satisfied, value q in object $\langle MEM, q \rangle$ is decreased by one. In addition, if all clauses are checked, then object $\langle CHECK \rangle$ is created. In the case where all clauses are satisfied, objects $\langle MEM, 0 \rangle$ and $\langle CHECK \rangle$ are in the same membrane, and object $\langle TRUE \rangle$ is then sent to the outer membrane.

On the other hand, object $\langle FALSE, 2^k\rangle$ is created if there is a clause that cannot be satisfied. (Here, k is the number of unassigned literals in object $\langle LIT, k\rangle$.) The object is used in Step 6 to detect the unsatisfiability of the formula.

Otherwise, objects that are obtained at the end of Step 1 are created again, and the objects trigger Step 2.

Step 6

In Step 6, one of the final objects, $\langle TRUE\rangle$ or $\langle FALSE\rangle$, is sent from the outer membrane if objects indicating "TRUE" or "FALSE" are sent out from the inner membrane. Step 6 is executed by applying the following evolution rules:

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,6} \;=\; & \{[\langle TRUE\rangle]_1 \rightarrow []_1\langle TRUE\rangle\} \\ & \cup\{\langle FALSE, 2^k\rangle\langle FALSE, 2^k\rangle \rightarrow \langle FALSE, 2^{k+1}\rangle \mid 0 \le k \le n-1\} \\ & \cup\{[\langle FALSE, 2^n\rangle]_1 \rightarrow []_1\langle FALSE\rangle\} \end{aligned}$$

The computation using the above evolution rules is executed as follows. Object $\langle FALSE, 2^k\rangle$ is combined until the object is merged into $\langle FALSE, 2^n\rangle$. In the case where $\langle FALSE, 2^n\rangle$ or $\langle TRUE\rangle$ is in the outer membrane, an output is determined, and $\langle TRUE\rangle$ or $\langle FALSE\rangle$ is sent out from the outer membrane.

We now summarize the asynchronous P system Π_{DPLL_SAT} as follows:

$$\Pi_{DPLL\_SAT} = (O, \mu, \omega_0, \omega_1, R_0, R_1, i_{in}, i_{out})$$

$$\begin{aligned} O \;=\; & \{\langle X_{i,j}, V\rangle \mid 1 \le i \le n, 1 \le j \le m, V \in \{N, 0, 1\}\} \\ & \cup\{\langle M_{i,j}, l\rangle \mid 1 \le i \le n, 1 \le j \le m, 0 \le l \le n\} \\ & \cup\{\langle F_j, i, V\rangle \mid 1 \le i \le n, 1 \le j \le m, V \in \{0, 1\}\} \\ & \cup\{\langle PUR, V\rangle \mid V \in \{N, 0, 1\}\} \\ & \cup\{\langle A_i, V\rangle \mid 0 \le i \le n, V \in \{N, 0, 1\}\} \\ & \cup\{\langle LIT, i\rangle \mid 0 \le i \le n\} \\ & \cup\{\langle MEM, j\rangle \mid 0 \le j \le m\} \\ & \cup\{\langle P_{i,j}\rangle \mid 1 \le i \le n+1, 1 \le j \le m\} \\ & \cup\{\langle N_j\rangle \mid 1 \le j \le m+1\} \\ & \cup\{\langle K_{i,j}\rangle \mid 1 \le i \le n+1, 1 \le j \le m+1\} \\ & \cup\{\langle C_{i,j}, i\rangle \mid 1 \le i \le n+1, 1 \le j \le m+1\} \\ & \cup\{\langle S_i\rangle \mid 1 \le i \le n\} \\ & \cup\{\langle FALSE, 2^i\rangle \mid 0 \le i \le n\} \\ & \cup\{\langle CHECK\rangle\langle TRUE\rangle\} \\ \mu \;=\; & [\,[\,]_2\,]_1 \\ \omega_1 \;=\; & \omega_2 \;=\; \phi \\ R_1 \;=\; & R_{1,1} \cup R_{1,6} \\ R_2 \;=\; & R_{2,1} \cup R_{2,2,1} \cup R_{2,2,2} \cup R_{2,3,1} \cup R_{2,3,2} \cup R_{2,4} \cup R_{2,5,1} \cup R_{2,5,2} \\ i_{in} \;=\; & i_{out} = 1 \end{aligned}$$

3.4 An example of execution of the proposed P system

An example of execution of the proposed P system is shown in Figure 1 to Figure 8. An input formula of the example is as follows.

$$L = (X_1) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_4) \wedge (\neg X_2 \vee X_3 \vee \neg X_4)$$

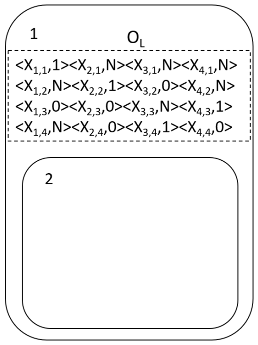

Figure 1: Initial state

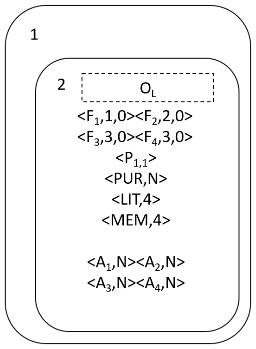

Figure 2: Step 1

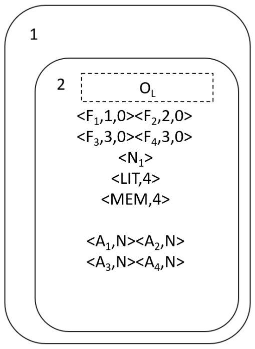

Figure 3: Step 2

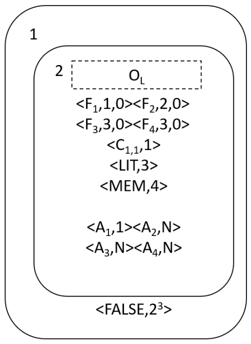

Figure 4: Step 3

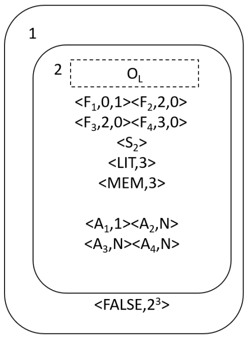

Figure 5: Step 5

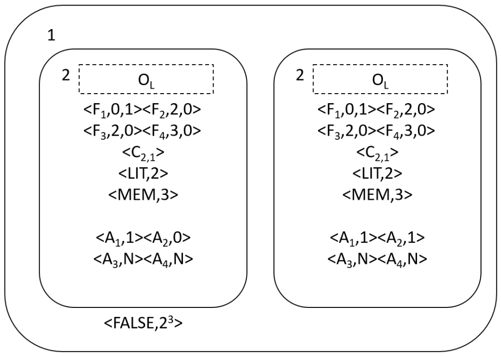

Figure 6: Step 4

Figure 1 illustrates an initial state of the P system. A set of an input object O_L is given in membrane 1. Figure 2 illustrates an execution of Step 1. The set of objects O_L is moved into membrane 2 by applying evolution rules $R_{1,1}$ and $R_{2,1}$.

Figure 3 illustrates an execution of Step 2. Since a pure literal rule cannot be applied for the input, the execution is moved to Step 3, which is illustrated in Figure 4. In Step 3, one-literal-rule can be applied for X_1 in the input, and X_1 is set to 1. In this case, object $\langle FALSE, 2^3\rangle$ is generated, and the object is moved from membrane 1 to membrane 2. Since the pure literal rule is applied in Step 3, Step 4 is skipped, and the execution is moved to Step 5.

Figure 5 illustrates an execution of Step 5. At this time, the Boolean formula could be given as follows because of the assignment for X_1.

$$L = (X_2 \vee \neg X_3) \wedge (\neg X_2 \vee X_4) \wedge (\neg X_2 \vee X_3 \vee \neg X_4)$$

In the figure, object $\langle F_1, 0, 1\rangle$ means that the first clause is satisfied. In addition, the number of literals in the third clause is changed from three to two because $\neg X_1$ is in the third clause, and object $\langle F_3, 3, 0\rangle$ is changed to $\langle F_3, 2, 0\rangle$. Then, an object that triggers Step 2 is generated because all clauses are not satisfied. After that, a splitting rule is applied in Step 4 since none of pure literal rule and one-literal rule can be applied. In Figure 6, which illustrates an execution of Step 4, the inner membrane is divided into two membranes, and X_2 is set to 1 in the left inner membrane, and X_2 is set to 0 in the right inner membrane.

Next, the execution is moved to Step 5, and then, Step 2. Figure 7 illustrates an execution in Step 2. In the left inner membrane, the input formula is now $L = (\neg X_3)$, and X_3 set to 0 applying pure literal rule. In the right inner membrane, the input formula is now $L = (X_4) \wedge (X_3 \vee \neg X_4)$, and X_3 set to 1 applying pure literal rule. Then, an execution is moved to Step 5.

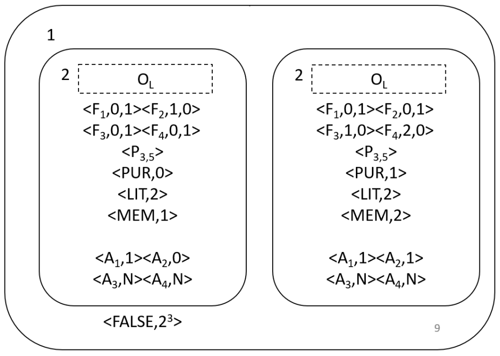

Figure 7: Step 2 (2nd)

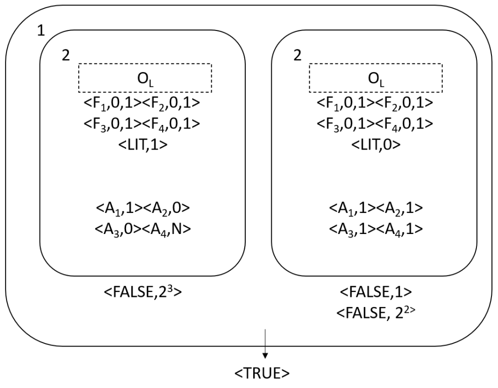

Figure 8: Step 6

In Step 5, all clauses are satisfied in the left inner membrane, and $\langle TRUE\rangle$ is generated, and then, the the object is outputted in Step 6, which is illustrated in Figure 8. In this case, object $\langle TRUE\rangle$ is send out from the outer membrane.

3.5 Complexity of the P system

The number of membrane is 2^n at worst.

In Step 1, since $O(mn)$ objects move sequentially, Step 1 are executed in $O(mn)$ parallel steps or $O(mn)$ sequential steps.

In Step 2, since each variable in each clause are checked, Step 2 are executed in $O(mn)$ parallel steps or $O(mn2^n)$ sequential steps. Evolution rules are $O(mn^2)$.

In Step 3, same as Step 2, Step 3 are executed in $O(mn)$ parallel steps or $O(mn2^n)$ sequential steps.

In Step 4, since a variable which is not assigned are checked, Step 4 are executed in $O(n)$ parallel steps or $O(n2^n)$ sequential steps. Evolution rules are $O(n^2)$.

In Step 5, since one variable in each membrane are checked, Step 5 are executed in $O(m)$ parallel steps or $O(m2^n)$ sequential steps. Evolution rules are $O(m^2n^2)$.

Steps 2 through 5 are repeated at worst n times.

In Step 6, Step 6 are executed in $O(1)$ parallel steps or $O(1)$ sequential steps. Evolution rules are $O(n)$.

From the above, we obtain the following theorem for the complexity of the proposed asynchronous P system Π_{DPLL_SAT}.

Theorem 1 *The asynchronous P system Π_{DPLL_SAT} solves the satisfiability problem (SAT) with n variables and m clauses and operates in $O(mn^2)$ parallel steps or $O(mn^2 2^n)$ sequential steps using $O(m^2n^2)$ types of objects and $O(2^n)$ membranes.* □

Since SAT is a computationally hard problem, the worst case complexity for the proposed P system is still exponential. We evaluate the validity of the proposed P system using experimental simulations in the next section.

4 Experimental simulations

For the experimental simulations, we use our original simulator for the asynchronous P system. The simulator is built using Python 3 and is executed on CentOS 7. The input formula is randomly created for a given number of variables n and clauses m.

We first compare the number of membranes for the proposed P system with two existing P systems, which are a P system with exhaustive search [10] and a P system with branch and bound [3].

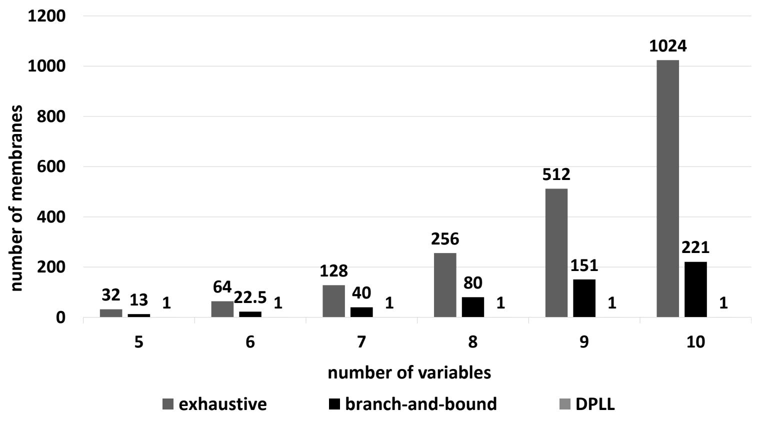

Figure 9: Number of membranes on known P systems and the proposed P system

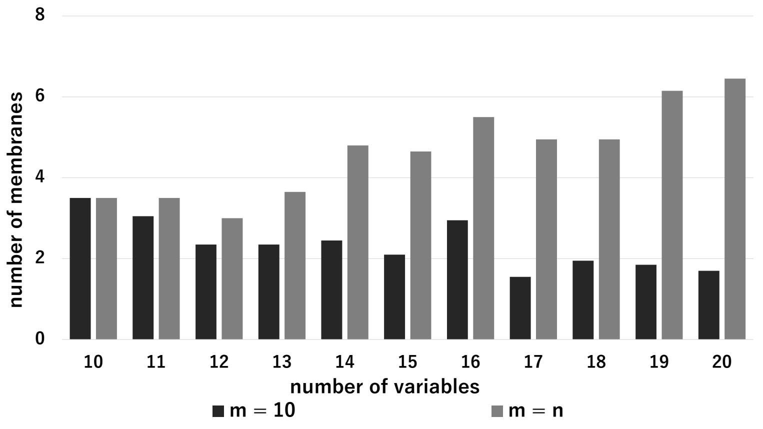

Figure 10: Number of membranes on the proposed P system for a number of variables between 10 and 20

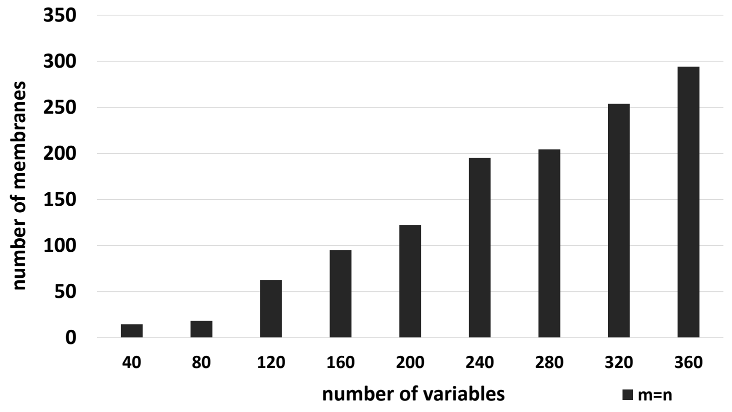

Figure 11: Number of membranes on the proposed P system for larger inputs

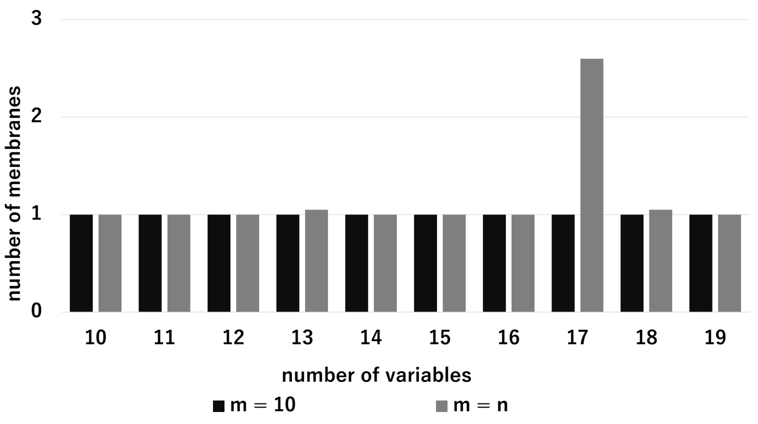

Figure 12: Number of membranes for 3-SAT

The simulation is executed for a small number of variables because the P system with exhaustive search [10] cannot be executed for a large number of membranes. Each clause consists of one to three randomly selected literals.

Figure 9 shows the number of membranes in the case where the number of variables n is between 5 and 10 and the number of clauses is $m = 3$. Each number of membranes is obtained as an average of 50 trials. In this case, the number of membranes used in the P system in [10] is exponential, and the number of membranes used in the P system in [3] is better than the result for the P system described in [10]. On the other hand, the number of membranes used in the proposed P system is always one, regardless of the numbers of variables. In other words, the splitting rule is not applied in the case where the number of variables is less than 10, although the worst number of membranes is $O(2^n)$.

We next evaluate the number of membranes of the proposed P system for larger numbers of variables. We assume that the number of variables n is between 10 and 20, and we also assume that the number of clauses m in two cases ($m = 10$ or $m = n$). Each number of membranes is obtained as an average of 25 trials. The number of literals in each clause is set to a randomly selected number between 1 and n because an increase in the number of literals in each clause causes an increase in the number of applications of the splitting rule.

Figure 10 shows the number of membranes for these two cases. In the case where $m = 10$, the number of membranes decreases with increasing number of variables. The results implies that the pure literal rule and the single literal rule can be applied increasingly according to the number of variables.

In the case where $m = n$, the number of membranes increases with the number of variables. The results imply that even if the one-literal rule and the pure literal rule can be applied more often, the number of clauses that cannot be applied to the rules increases with increasing number of clauses. If the one-literal rule and the pure literal rule are applied in early steps, the number of clauses can be reduced; otherwise, the number of membranes increases significantly. In other words, an increase in the number of clauses affects the number of membranes more than an increase in the number of variables.

Figure 11 shows the number of membranes of the proposed P system for even larger inputs. The number of variables is between 40 and 360, and the number of clauses is set to be the same as the number of variables, that is, $m = n$. Each number of membranes is obtained as an average of 20 trials. In this case, the number of membranes seems to increase linearly as the number of variables increases. In other words, we prevent an exponential increase of the number of membranes. The fact shows that the more variables are in the input, the more membranes can be reduced effectively in proposed P system.

Finally, we assume that the input is 3-SAT[1]. Figure 12 shows the results of the simulation for the proposed P system in this case, and each number of membranes is obtained as an average of 25 trials. The results show that the number of membranes is approximately constant, and the splitting rule is rarely applied in the case of 3-SAT. The reason of the peak for $m = n$ with 17 variables is that there is a rare input such that the number of membrane is not 1, and the case is in 25 trials in case of the 17 variables.

5 Conclusions

In the present paper, we proposed an asynchronous P system with a DPLL algorithm for solving SAT. This system reduces the number of membranes by eliminating useless variable assignments.

We examined the proposed P system and existing P systems in a simulation environment. The results show that the number of membranes used in the proposed P system is significantly smaller than the numbers of membranes used in existing P systems.

In our future research, we intend to consider the reduction of the number of membranes for other computationally hard problems.

Acknowledgment

This research was supported in part by JSPS KAKENHI through a Grant-in-Aid for Scientific Research (C) (No. 20K11681).

References

[1] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[2] R. Freund. Asynchronous P systems and P systems working in the sequential mode. *WMC'04 Proceedings of the 5th international conference on Membrane Computing*, 3365:36–62, 2005.

[3] Y. Jimen and A. Fujiwara. Asynchronous P systems for solving the satisfiability problem. *International Journal of Networking and Computing*, 8(2):141–152, 2018.

[4] A. Leporati and C. Zandron. P systems with input in binary form. *International Journal of Foundations of Computer Science*, 17:127–146, 2006.

[5] T. Murakawa and A. Fujiwara. Asynchronous P system for arithmetic operations and factorization. *Proceedings of 3rd International Workshop on Parallel and Distributed Algorithms and Applications*, 2011.

[6] Y. Nakano and A. Fujiwara. An asynchronous P system with branch and bound for solving the knapsack problem. In *Workshop on Parallel and Distributed Algorithms and Applications*, pages 242–248, 2020.

[7] L. Pan and A. Alhazov. Solving HPP and SAT by P systems with active membranes and separation rules. *Acta Informatica*, 43(2):131–145, 2006.

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

[9] G. Păun. P system with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–95, 2001.

[1]In 3-SAT, at most three literals exist in each clause.

[10] H. Tagawa and A. Fujiwara. Solving SAT and Hamiltonian cycle problem using asynchronous p systems. *IEICE Transactions on Information and Systems (Special section on Foundations of Computer Science)*, E95-D(3), 2012.

[11] K. Tanaka and A. Fujiwara. Asynchronous P systems for hard graph problems. *International Journal of Networking and Computing*, 4(1):2–22, 2014.

[12] T. Tateishi and A. Fujiwara. Logic and arithmetic operations with a constant number of steps in membrane computing. *International Journal of Foundations of Computer Science*, 22(3):547–564, 2011.

[13] K. Umetsu and A. Fujiwara. P systems with branch and bound for solving two hard graph problems. *International Journal of Networking and Computing*, 10(2):159–173, 2020.

[14] C. Zandron, G. Rozenberg, and G. Mauri. Solving NP-complete problems using P systems with active membranes. *Proceedings of the Second International Conference on Uncoventional Models of Computation*, pages 289–301, 2000.