Asynchronous membrane computing for the compare-and-exchange and sorting

Yutaka Nishida Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan
Email: k232060y@mail.kyutech.jp, fujiwara@cse.kyutech.ac.jp

***Abstract*—Recently, membrane computing, which is a computational model based on cell activity, has considerable attention as one of new paradigms of computations. In the membrane computing, the asynchronous parallelism must be considered to make the membrane computing more realistic.**

In the present paper, we propose asynchronous P systems that execute a compare-and-exchange operation and sorting. We first propose an asynchronous P system for the compare-and-exchange operation of two binary numbers of m bits. The P system works in $O(m)$ steps by using $O(m)$ types of objects, a constant number of membranes and evolution rules of size $O(m)$. We next propose an asynchronous P system for sorting of n binary numbers of m bits by using the above asynchronous P system as a sub-system. The P system works in $O(mn^2)$ steps by using $O(mn)$ types of objects, a constant number of membranes, and evolution rules of size $O(mn)$.

I. Introduction

A number of next-generation computing paradigms have been considered due to limitation of silicon-based computational hardware. As an example of the computing paradigms, natural computing, which works using natural materials for computation, has considerable attention. A membrane computing, which is a computational model inspired by the structures and behaviors of living cells, is a representative of the natural computing.

A basic feature of the membrane computing was introduced by in [1] as a P system. The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. Each object evolves according to evolution rules associated with a membrane in which the object is contained.

The P system and most variants have been proved to be universal [2], and several P systems have been proposed for solving NP problems [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] since a exponential number of membranes can be created in a polynomial number of steps on the P system. In addition, P systems for basic operations, such as logic or arithmetic operations, have been proposed in [13], [14] to apply membrane computing in a wide range problems.

However, synchronous application of evolution rules is assumed on the above P systems with the maximal parallelism, which is a main feature of the P systems. The maximal parallelism means that all applicable rules in all membranes are applied synchronously.

On the other hand, there is obvious asynchronous parallelism in the cell biochemistry. The asynchronous parallelism means that all objects may react on rules with different speed, and evolution rules are applied to objects independently. Since all objects in a living cell basically works in asynchronous manner, the asynchronous parallelism must be considered to make P system more realistic model.

For considering asynchronous parallelism, a number of P systems have been proposed in [15], [16], [17], [16], [18]. As an example, two asynchronous P systems [17] have been proposed for solving SAT and Hamiltonian cycle problem, and a number of P systems [18] have been proposed for graph problems. The P systems solve NP problems in a polynomial number of parallel steps. In addition, another asynchronous P system [16] has been proposed for computing arithmetic operations and factorization.

As complexity of the asynchronous P system, we consider two kinds of numbers, which are a number of sequential steps and a number of parallel step. The numbers of sequential steps is a number of executed steps in case that rules are applied sequentially, and the number of parallel steps is a number of executed steps with maximal parallelism.

In the present paper, we propose asynchronous P systems that executes a compare-and-exchange operation and sorting, which are basic operations for computation. We first propose an asynchronous P system for the compare-and-exchange operation of two binary numbers of m bits. The P system works in $O(m)$ sequential and parallel steps by using $O(m)$ types of objects, a constant number of membranes and evolution rules of size $O(m)$.

We next propose an asynchronous P system for sorting of n binary numbers of m bits by using the above asynchronous P system as a sub-system. The P system works in $O(mn^2)$ sequential and parallel steps by using $O(mn)$ types of objects, a constant number of membranes, and evolution rules of size $O(mn)$.

II. Preliminaries

A. Computational model for membrane computing

Several models have been proposed for membrane computing. We briefly introduce a basic model of the P system in this subsection. The P system consists mainly of membranes and objects. A membrane is a computing cell, in which

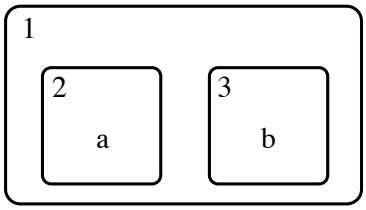

Fig. 1. An example of membrane structure

independent computation is executed, and may contain objects and other membranes. In other words, the membranes form nested structures. In the present paper, each membrane is denoted by using a pair of square brackets, and the number on the right-hand side of each right-hand bracket denotes the label of the corresponding membrane. An object in the P system is a memory cell, in which each data is stored, and can divide, dissolve, and pass through membranes. In the present paper, each object is denoted by finite strings over a given alphabet, and is contained in one of the membranes.

For example, $[[a]_2[b]_3]_1$ and Figure 1 denote the same membrane structure that consists of three membranes. The membrane labeled 1 contains two membranes labeled 2 and 3, and the two membranes contain objects a and b, respectively.

Computation of P systems is executed according to evolution rules, which are defined as rewriting rules for membranes and objects. All objects and membranes are transformed in parallel according to applicable evolution rules. If no evolution rule is applicable for objects, the system ceases computation.

Now, we formally define a P system and the sets used in the system as follows.

$$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m, R_1, R_2, \cdots, R_m)$$

O: O is the set of all objects used in the system.

ω_i: Each ω_i is a set of objects initially contained only in the membrane labeled i.

R_i: Each R_i is a set of evolution rules that are applicable to objects in the membrane labeled i.

In the present paper, we assume that input objects are given from the outside region into the outermost membrane, and computation is started by applying evolution rules. We also assume that output objects are sent from the outermost membrane to the outside region. In membrane computing, several types of rules are proposed. In the present paper, we consider five basic rules of the following forms.

(1) Object evolution rule: $[\ a\ ]_h \rightarrow [\ b\ ]_h$
where $h \in H$ and $a, b \in O$. Using the rule, an object a evolves into another object b. (We omit the brackets in each evolution rule for cases that a corresponding membrane is obvious.)

(2) Send-in communication rule: $a[\ ]_h \rightarrow [\ b\ ]_h$
where $h \in H$, and $a, b \in O$. Using the rule, an object a is sent into the membrane, and can evolve into another object b.

(3) Send-out communication rule: $[\ a\ ]_h \rightarrow [\ ]_h b$
where $h \in H$, and $a, b \in O$. Using the rule, an object a is sent out of the membrane, and can evolve into another object b.

(4) Dissolution rule: $[\ a\ ]_h \rightarrow b$
where $h \in H$, and $a, b \in O$. Using the rule, the membrane, which contains object a, is dissolved, and the object can evolve into another object b. (The outermost membrane cannot be dissolved.)

(5) Division rule: $[\ a\ ]_h \rightarrow [\ b\ ]_h[\ c\ ]_h$
where $h \in H$, and $a, b \in O$. Using the rule, the membrane, which contains object a, is divided into two membranes that contain objects b and c, respectively.

We assume that each of the above rules is applied in a constant number of biological steps. In the following sections, we consider the number of steps executed in a P system as the complexity of the P system.

B. Maximal parallelism and asynchronous parallelism

In the standard model in membrane computing, which is a P system with maximal parallelism, all of the above rules are applied in a non-deterministic maximally parallel manner. In one step of computation of the P system, each object is evolved according to one of applicable rules. (In case there are several possibilities, one of the applicable rules is non-deterministically chosen.) All object, for which no rules applicable, remain unchanged to the next step. In other words, all applicable rules are applied in parallel in each step of computation.

On the other hand, we propose asynchronous P systems, which assume that evolution rules are applied in fully asynchronous manner. In the asynchronous P systems, any number of applicable evolution rules is applied in each step of computation. In the other words, the asynchronous P system can be executed sequentially, and also can be executed in maximal parallel manner.

The reason why we assume the asynchronous parallelism in this paper is based on the fact that every living cell acts independently and asynchronously. Since the standard P system ignores the asynchronous feature of living cells, the asynchronous P system is a more realistic computation model for cell activities.

We now show an example for difference between the P system with maximal parallelism and the asynchronous P system. We define P system Π and the sets used in the system as follows.

$$\Pi = (O, \mu, \omega_1, R_1)$$

- $O = \{a, b, c, d, e\}$
- $\mu = [\ ]_1$
- $\omega_1 = \phi$
- $R_1 = \{a \rightarrow b, bc \rightarrow d, c \rightarrow e\}$

We now show an example of the computation of the P system Π. Let us assume that input objects aac are given into the membrane from the outside region.

We first consider a computation of Π on the standard P system. In the initial state, the applicable rules are $a \rightarrow b$ and

$c \to e$, and the two rules are applied in parallel with maximal parallelism. Then, input objects are evolved into bbe after the first computation step in the P system. Since the object bbe cannot be evolved using evolution rules in R_1, the computation on the P system is halted.

We next consider a computation of Π on the asynchronous P system. In the initial state of the asynchronous P system, the applicable rules are $a \to b$ and $c \to e$, and the two rules are applied asynchronously. Then, the input objects aac can be evolved into bbe, abe, bbc, abc or aae in the first step of the computations. In this case, objects bbc and abc can be evolved into bd and ad in the second step of the computation, respectively.

Therefore, a number of executions are possible in the asynchronous P system, and the evolution rules in the standard P system, which assumes a maximal parallel manner, may not work in an asynchronous parallel manner.

In the asynchronous P system, all evolution rules can be applied completely in parallel, which is the same as the conventional P system, or all evolution rules can be applied sequentially. We define the number of steps executed in the asynchronous P system in the maximal parallel manner as the number of parallel steps. We also define the number of steps in the case that the applicable evolution rules are applied sequentially as the number of sequential steps. The numbers of parallel and sequential steps indicate the best and worst case complexities for the asynchronous P system. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

C. Representation of binary numbers with objects

In this subsection, we describe a unified representation of a binary number with objects. The representation is similar to the binary notation of [14], and one object corresponds to one bit of a binary number. Therefore, we use $O(mn)$ objects to denote n binary numbers of m bits. In addition, the representation enables the addressing feature, i.e., each binary number is stored in a given address.

Let $V_{i,m-1}, V_{i,m-2}, \cdots, V_{i,0}$ be m Boolean values stored in address i. In case that the values denote a non-negative integer V_i, the following expression holds.

$$V_i = \sum_{j=0}^{m-1} V_{i,j} \times 2^j$$

.

We use the following m objects to denote a binary number of m bits. In the objects, A_i and B_j denote the address and the bit position, respectively, in which each value is stored.

$$\langle A_i, B_{m-1}, V_{i,m-1}\rangle, \langle A_i, B_{m-2}, V_{i,m-2}\rangle, \cdots, \langle A_i, B_0, V_{i,0}\rangle$$

The above objects are referred to as memory objects. For example, the following four memory objects denote a binary number 1000, which is stored in address 1.

$$\langle A_1, B_3, 1\rangle, \langle A_1, B_2, 0\rangle, \langle A_1, B_1, 0\rangle, \langle A_1, B_0, 0\rangle$$

III. Compare-and-exchange

In this section, we present an asynchronous P system for the compare-and-exchange operation of two binary numbers of m bits. The input of the compare-and-exchange operation is a pair of two values p, q, and the output of the operation is also a pair of two values x, y such that $x = \min\{p, q\}$ and $y = \max\{p, q\}$. We first explain an input and an output of the compare-and-exchange operation for the P system, and then, show an outline and details of the P system with an example. Finally, we discuss time complexity of the proposed P system.
‘

A. Input and output

An input and an output of the compare-and-exchange operation are expressed using memory objects described in Section 2.

We assume that two input binary numbers of m bits is stored in addresses p and q. The following two sets of objects are given as an input in the outermost membrane.

$$\langle A_p, B_{m-1}, V_{p,m-1}\rangle, \langle A_p, B_{m-2}, V_{p,m-2}\rangle, \cdots, \langle A_p, B_0, V_{p,0}\rangle$$
$$\langle A_q, B_{m-1}, V_{q,m-1}\rangle, \langle A_q, B_{m-2}, V_{q,m-2}\rangle, \cdots, \langle A_q, B_0, V_{q,0}\rangle$$

An output of the compare-and-exchange operation, which is a pair of two binary numbers stored in addresses x and y, is also given as sets of memory objects as follows.

$$\langle A_x, B_{m-1}, V_{x,m-1}\rangle, \langle A_x, B_{m-2}, V_{x,m-2}\rangle, \cdots, \langle A_x, B_0, V_{x,0}\rangle$$
$$\langle A_y, B_{m-1}, V_{y,m-1}\rangle, \langle A_y, B_{m-2}, V_{y,m-2}\rangle, \cdots, \langle A_y, B_0, V_{y,0}\rangle$$

B. An asynchronous P system for the compare-and-exchange

We first explain an overview of the asynchronous P system for the compare-and-exchange operation. The membrane structure used in the computation is the outermost membrane only such that $[\ ]_1$.

The computation of the P system consists of the following 3 steps.

- Step 1: Find the most significant bit, which is the left-most bit position such that Boolean values of two binary numbers p and q differs. Then, compute the relation between p and q from the most significant bit.
- Step 2: In case that $p \geq q$, copy all Boolean values of p and q to memory objects that denotes y and x, respectively. In the other case, copy all Boolean values of p and q to memory objects that denotes x and y, respectively.
- Step 3: Send out memory objects that denote x and y from the outermost membrane.

We now explain outline of each step of the computation. In Step 1, the most significant bit is searched from the higher bit to the lower bit applying the following two sets of evolution rules.

$$\begin{aligned} R_{1,1,1} \quad = \quad & \{\langle A_p, B_i, V\rangle\langle A_q, B_i, V\rangle\langle CB, i\rangle \\ & \to \langle A_p, B_i, V\rangle\langle A_q, B_i, V\rangle\langle CB, i-1\rangle \\ & \mid V \in \{0,1\}, 1 \leq i \leq m-1\} \end{aligned}$$

$$
\begin{aligned}
R_{1,1,2} \;=\; & \{\langle A_p, B_i, 1\rangle\langle A_q, B_i, 0\rangle\langle CB, i\rangle \\
& \quad \to \langle A_p, B_i, 1\rangle\langle A_q, B_i, 0\rangle\langle GTE\rangle \\
& \quad \mid 0 \le i \le m-1\} \\
& \cup\{\langle A_p, B_i, 0\rangle\langle A_q, B_i, 1\rangle\langle CB, i\rangle \\
& \quad \to \langle A_p, B_i, 0\rangle\langle A_q, B_i, 1\rangle\langle LT\rangle \\
& \quad \mid 0 \le i \le m-1\} \\
& \cup\{\langle A_p, B_0, V\rangle\langle A_q, B_0, V\rangle\langle CB, 0\rangle \\
& \quad \to \langle A_p, B_0, V\rangle\langle A_q, B_0, V\rangle\langle GTE\rangle \\
& \quad \mid V \in \{0,1\}\}
\end{aligned}
$$

In the above evolution rules, object $\langle CB, i\rangle$ denotes current bit position for comparison of two Boolean values. In case that two Boolean values are equal, except for the lowest bits, evolution rules in $R_{1,1,1}$ is applied, and the comparison is moved to the lower bit position. In the other case, one of objects $\langle GTE\rangle$ and $\langle LT\rangle$, which denote "greater than or equal to" and "less than" respectively, is created according to evolution rules in $R_{1,1,2}$.

In Step 2, Boolean values that denote p and q are copied to memory objects that denote x and y according to the results of the comparison in Step 1. Step 2 is executed applying the following set of evolution rule.

$$
\begin{aligned}
R_{1,2} \;=\; & \{\langle GTE\rangle \to \langle EX, m-1\rangle, \langle LT\rangle \to \langle CP, m-1\rangle\} \\
& \cup\{\langle A_p, B_i, V_p\rangle\langle A_q, B_i, V_q\rangle\langle EX, i\rangle \\
& \quad \to \langle A_x, B_i, V_q\rangle\langle A_y, B_i, V_p\rangle\langle EX, i-1\rangle, \\
& \quad \langle A_p, B_i, V_p\rangle\langle A_q, B_i, V_q\rangle\langle CP, i\rangle \\
& \quad \to \langle A_x, B_i, V_p\rangle\langle A_y, B_i, V_q\rangle\langle CP, i-1\rangle \\
& \quad \mid V_p, V_q \in \{0,1\}, 1 \le i \le m-1\} \\
& \cup\{\langle EX, -1\rangle \to \langle CB, m-1\rangle, \\
& \quad \langle CP, -1\rangle \to \langle CB, m-1\rangle\}
\end{aligned}
$$

In the above $R_{1,2}$, object $\langle EX, i\rangle$ executes exchange of two input values p and q, i.e. the object copies values p and q to y and x from the higher bit to the lower bit. On the other hand, object $\langle CP, i\rangle$ similarly copies two input values p and q to x and y. At the end of Step 2, object $\langle CB, m-1\rangle$ is created for initializing the object for the next compare-and-exchange operation.

In Step 3, memory objects, which denote x and y, are sent out from the outermost membrane applying the following set of send-out communication rules.

$$
\begin{aligned}
R_{1,3} \;=\; & \{[\langle A_x, B_i, V_x\rangle\langle A_y, B_i, V_y\rangle]_1 \\
& \quad \to [\,]_1\ \langle A_x, B_i, V_x\rangle\langle A_y, B_i, V_y\rangle \\
& \quad \mid V_x, V_y \in \{0,1\}, 0 \le i \le m-1\}
\end{aligned}
$$

Note that Step 3 may be executed before finishing Step 2 because we assume the asynchronous P system. In any execution of the P system, the P system output correct results at the end of computation because sets of evolution rules is designed to be applied sequentially.

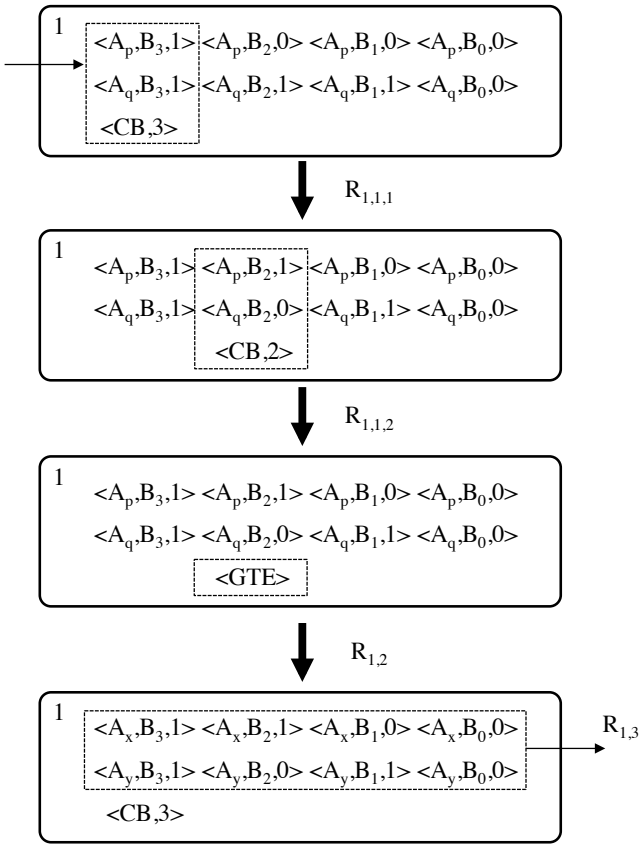

Fig. 2. An example of execution of Π_{ce}

We now formally define P system π_{ce} that executes the compare-and-exchange operation for two binary numbers of m bits.

$$\Pi_{ce} = (O, \mu, \omega_1, R_1)$$

$$
\begin{aligned}
O \;=\; & \{\langle A_p, B_i, V_p\rangle, \langle A_q, B_i, V_q\rangle, \\
& \quad \langle A_x, B_i, V_x\rangle, \langle A_y, B_i, V_y\rangle \\
& \quad \mid V_p, V_q, V_x, V_y \in \{0,1\}, 0 \le i \le m-1\} \\
& \cup\{\langle CB, i\rangle \mid 0 \le i \le m-1\} \\
& \cup\{\langle EX, i\rangle, \langle CP, i\rangle \mid -1 \le i \le m-1\} \\
& \cup\{\langle GTE\rangle, \langle LT\rangle\} \\
\mu \;=\; & [\,]_1 \\
\omega_1 \;=\; & \{\langle CB, m-1\rangle\} \\
R_1 \;=\; & R_{1,1,1} \cup R_{1,1,2} \cup R_{1,2} \cup R_{1,3}
\end{aligned}
$$

Figure 2 illustrates an example of an execution of the P system Π_{ce}. In the example, two input binary numbers are $p = 1100$ and $q = 1010$, and the memory objects that denote p and q are given from the outside region into the P system. Then, the comparison for the first bit is executed applying $R_{1,1,1}$, and the comparison is moved the the next bit.

In the next step, a set of rule $R_{1,1,2}$ is applied, and object $\langle GTE\rangle$ is created in the membrane, and the object enables

$R_{1,2}$. After application of $R_{1,2}$, exchanged values are sent out from the outermost membrane applying $R_{1,3}$.

C. Complexity of the P system

We now consider complexity of the proposed P system Π_{ce}. Both of the number of sequential and parallel steps in Step 1 and Step 2 are $O(m)$ because the steps are executed sequentially. The number of sequential and parallel steps in Step 3 is $O(m)$ and $O(1)$, respectively, because the step can executed in parallel.

Since the number of types of objects in the P system is $O(m)$ and the number of kinds of evolution rules is also $O(m)$, we obtain the following theorem for Π_{ce}.

Theorem 1: An asynchronous P system Π_{ce}, which executes compare-and-exchange operation of two binary numbers of m bits, works in in $O(m)$ sequential and parallel steps using $O(m)$ types of objects and evolution rules of size $O(m)$. □

IV. Sorting

We next show an asynchronous P system for sorting n binary numbers of m bits. An idea of the P system is based on the odd-even transposition sort [19], and the P system Π_{ce}, which is described in previous section, is used as a sub-system. We first explain an input and an output of sorting for our P system, and then, show an outline and details of our P system. Finally, we discuss time complexity of the proposed P system.

A. Input and output

We assume that input of sorting is n binary numbers $V_0, V_1, \cdots, V_{n-1}$, and also assume that the numbers are stored in addresses $X_0, X_1, \cdots X_{n-1}$. The binary numbers are given as a set of memory objects given below.

$$\begin{aligned}
&\langle X_0, B_{m-1}, V_{0,m-1}\rangle\langle X_0, B_{m-2}, V_{0,m-2}\rangle\cdots\langle X_0, B_0, V_{0,0}\rangle\\
&\langle X_1, B_{m-1}, V_{1,m-1}\rangle\langle X_1, B_{m-2}, V_{1,m-2}\rangle\cdots\langle X_1, B_0, V_{1,0}\rangle\\
&\qquad\vdots\\
&\langle X_{n-1}, B_{m-1}, X_{n-1,m-1}\rangle\langle X_{n-1}, B_{m-2}, X_{n-1,m-2}\rangle\cdots\\
&\qquad\cdots\langle A_{n-1}, B_0, V_{n-1,0}\rangle
\end{aligned}$$

We also assume that output of sorting is a set of binary numbers stored in addresses $Y_0, Y_1, \cdots Y_{n-1}$. The memory objects for output are given below.

$$\begin{aligned}
&\langle Y_0, B_{m-1}, V_{0,m-1}\rangle\langle Y_0, B_{m-2}, V_{0,m-2}\rangle\cdots\langle Y_0, B_0, V_{0,0}\rangle\\
&\langle Y_1, B_{m-1}, V_{1,m-1}\rangle\langle Y_1, B_{m-2}, V_{1,m-2}\rangle\cdots\langle Y_1, B_0, V_{1,0}\rangle\\
&\qquad\vdots\\
&\langle Y_{n-1}, B_{m-1}, V_{n-1,m-1}\rangle\langle Y_{n-1}, B_{m-2}, V_{n-1,m-2}\rangle\cdots\\
&\qquad\cdots\langle Y_{n-1}, B_0, V_{n-1,0}\rangle
\end{aligned}$$

B. An asynchronous P system for sorting

We first explain an overview of the asynchronous P system for sorting. The membrane structure used in the computation consists of three membranes such that $[\ [\ ]_{ce\_odd}\ [\ ]_{ce\_even}]_1$, where membranes *ce_odd* and *ce_even* is a P system for the compare-and-exchange operation for pairs of binary numbers. The computation of the P system consists of the following steps.

Step 1: Repeat the following two sub-steps, (1-1) and (1-2), $\frac{n}{2}$ times, and send out obtained memory objects from the outermost membrane.

(1-1) Send memory objects $V_0, V_1, \cdots V_{n-1}$ into membrane *ce_odd*. Then, execute the compare-and-exchange operation for $\frac{n}{2}$ pairs given below. (We call the the compare-and-exchange step *odd exchange step*.)

$$(V_{2i}, V_{2i+1}) \quad (0 \le i \le \frac{n}{2} - 1)$$

The above memory strands are sent out from the membrane after the odd exchange step.

(1-2) Send memory objects that denote $V_0, V_1, \cdots V_{n-1}$ into membrane *ce_even*. Then, execute the compare-and-exchange operation for $\frac{n}{2}$ pairs given below. (We call the compare-and-exchange step *even exchange step*.)

$$(V_{2i-1}, V_{2i}) \quad (1 \le i \le \frac{n}{2} - 1)$$

The above memory strands are also sent out from the membrane after the even exchange step.

We now explain an outline of each step of the computation. First of all, we consider two P systems, Π_{ce_odd} and Π_{ce_even}, which execute the odd and even exchange steps. Since each pair of compare-and-exchange operation is executed independently from the other pairs, Π_{ce_odd} and Π_{ce_even} are obtained by modifying Π_{ce}. We assume that these two P systems are two membranes *ce_odd* and *ce_even*.

We next explain the other steps for the P system. Since we assume an asynchronous P system, we must consider how to execute the above step synchronously, i.e., (1-1) and (1-2) must not be executed simultaneously.

In (1-1), input objects are moved into membrane *ce_odd*. This step is executed applying the following evolution rules.

$$\begin{aligned}
R_{1,1,1} \;=\; &\{\langle X_i, B_j, V\rangle\langle M_O, i, j\rangle[\ ]_{ce_odd}\\
&\quad\to[\langle X_i, B_j, V\rangle\langle M_O, i, j\rangle]_{ce_odd}\\
&\quad\mid V\in\{0,1\}, 0\le i\le n-1, 0\le j\le m-1\}\\
&\cup\{[\langle M_O, i, j\rangle]_{ce_odd}\to[\ ]_{ce_odd}\langle M_O, i, j+1\rangle\\
&\quad\mid 0\le i\le n-1, 0\le j\le m-2\}\\
&\cup\{[\langle M_O, i, m-1\rangle]_{ce_odd}\\
&\quad\to[\ ]_{ce_odd}\langle M_O, i+1, 0\rangle\\
&\quad\mid 0\le i\le n-1\}\\
&\cup\{\langle M_O, n, 0\rangle\langle C, k\rangle\to\langle M_E, 0, 0\rangle\langle C, k+1\rangle\\
&\quad\mid 0\le k\le n-1\}
\end{aligned}$$

In the above $R_{1,1,1}$, object $\langle A_i, B_j, V_{i,j}\rangle$ is moved into membrane *ce_odd* using object $\langle M_O, i, j\rangle$. The object $\langle M_O, i, j\rangle$ first moves memory objects stored in address 0, next moves memory objects stored in address 1, and so on. After all memory objects are moved into membrane *ce_odd*, the object is set to $\langle M_E, 0, 0\rangle$, which is an object used to move memory object into membrane *ce_even*. In addition, object

$\langle C, k\rangle$ is used to count the number of steps, and the object is incremented at the end of (1-1).

After moving memory objects into membrane *ce_odd*, the compare-and-exchange operation is executed in the membrane, and all memory objects are asynchronously sent out from membrane *ce_odd*.

In (1-2), input objects are moved into membrane *ce_odd* using the following $R_{1,1,2}$, which is similar to $R_{1,1,1}$.

$$\begin{aligned}
R_{1,1,2} &= \{\langle X_i, B_j, V\rangle\langle M_E, i, j\rangle[\ ]_{ce_even} \\
&\quad \to [\langle X_i, B_j, V\rangle\langle M_E, i, j\rangle]_{ce_even} \\
&\quad \mid V \in \{0,1\}, 0 \le i \le n-1, 0 \le j \le m-1\} \\
&\cup\{[\langle M_E, i, j\rangle]_{ce_even} \\
&\quad \to [\ ]_{ce_even}\langle M_E, i, j+1\rangle \\
&\quad \mid 0 \le i \le n-1, 0 \le j \le m-2\} \\
&\cup\{[\langle M_E, i, m-1\rangle]_{ce_even} \\
&\quad \to [\ ]_{ce_even}\langle M, i+1, 0\rangle \\
&\quad \mid 0 \le i \le n-1\} \\
&\cup\{\langle M_E, n, 0\rangle\langle C, k\rangle \to \langle M_O, 0, 0\rangle\langle C, k+1\rangle \\
&\quad \mid 0 \le k \le n-1\}
\end{aligned}$$

After $\frac{n}{2}$ times executions of (1-1) and (1-2), sorting is completed, and object $\langle M_O, 0, 0\rangle$ and $\langle C, n\rangle$is obtained. Then, all memory objects are sent out applying the following $R_{1,2}$.

$$\begin{aligned}
R_{1,2} &= \{\langle M_O, 0, 0\rangle\langle C, n\rangle \to \langle S, 0, 0\rangle\} \\
&\cup\{\langle X_i, B_j, V\rangle\langle S, i, j\rangle \\
&\quad \to \langle Y_i, B_j, V\rangle\langle S, i, j+1\rangle \\
&\quad \mid V \in \{0,1\}, 0 \le i \le n-1, 0 \le j \le m-2\} \\
&\cup\{\langle X_i, B_{m-1}, V\rangle\langle S, i, m-1\rangle \\
&\quad \to \langle Y_i, B_{m-1}, V\rangle\langle S, i+1, 0\rangle \\
&\quad \mid V \in \{0,1\}, 0 \le i \le n-1\} \\
&\cup\{[\langle Y_i, B_j, V\rangle]_1 \to [\ ]_1\langle Y_i, B_j, V\rangle \\
&\quad \mid V \in \{0,1\}, 0 \le i \le n-1, 0 \le j \le m-1\} \\
&\cup\{\langle S, n, 0\rangle \to \langle M_O, 0, 0, 1\rangle\}
\end{aligned}$$

We now formally define P system Π_{sort} that executes sorting for n binary numbers of m bits.

$$\Pi_{sort} = (O, \mu, \omega_1, \omega_{ce\_odd}, \omega_{ce\_even}, R_1, R_{ce\_odd}, R_{ce\_even})$$

$$\begin{aligned}
O &= \{\langle X_i, B_j, V\rangle, \langle Y_i, B_j, V\rangle \mid V \in \{0,1\}, \\
&\qquad 0 \le i \le n-1, 0 \le j \le m-1\} \\
&\quad \cup\{\langle M_O, i, j\rangle, \langle M_E, i, j\rangle \\
&\qquad \mid 0 \le i \le n, 0 \le j \le m-1\} \\
&\quad \cup\{\langle C, k\rangle \mid 0 \le k \le n\} \\
&\quad \cup\{\langle S, i, j\rangle \mid 0 \le i \le n, 0 \le j \le m-1\} \\
\mu &= [\ [\ ]_{ce_even}[\ ]_{ce_odd}]_1 \\
\omega_1 &= \{\langle M_O, 0, 0, 1\rangle\} \\
R_1 &= R_{1,1,1} \cup R_{1,1,2} \cup R_{1,2}
\end{aligned}$$

(ω_{ce_odd}, ω_{ce_even}, Π_{ce}. R_{ce_odd} and R_{ce_even} are omitted.)

C. Complexity of the P system

We now consider complexity of the proposed P system Π_{sort}. The numbers of parallel and sequential steps in (1-1) and (1-2) are $O(m)$ parallel steps and $O(mn)$ sequential steps because compare-and-exchange operations are executed for $\frac{n}{2}$ pairs of binary numbers. The other steps, which sequentially move memory objects, works in $O(nm)$. Since (1-1) and (1-2) is repeated $\frac{n}{2}$ times, time complexity of the P system is $O(mn^2)$ sequential and parallel steps.

The number of types of objects in the P system is $O(mn)$, and the number of kinds of evolution rules is also $O(mn)$. Therefore, we obtain the following theorem for Π_{sort}.

Theorem 2: An asynchronous P system Π_{sort}, which sorts n binary numbers of m bits, works in $O(mn^2)$ sequential and parallel steps using $O(mn)$ types of objects and evolution rules of size $O(mn)$. □

V. Conclusions

We proposed asynchronous P systems for the compare-and-exchange operation and sorting. The proposed P systems are fully asynchronous, i.e. any number of applicable rules may be applied in one step of the P systems. The first P system for the compare-and-exchange operation works in $O(m)$ sequential and parallel steps for two binary numbers of m bits, and the second P system for sorting works in $O(mn^2)$ sequential and parallel steps for n binary numbers of m bits. Although the number of steps is not small as well-known sorting algorithms, the proposed P system shows that the basic operations can be executed on the asynchronous P system.

As our future work, we are considering reduction of the numbers of parallel steps on the proposed asynchronous P systems.

Acknowledgments

This research was partially supported by JSPS KAKENHI, Grand-in-Aid for Scientific Research (C), 24500019.

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.
[2] ——, *Introduction to Membrane Computing*. Springer, 2006.
[3] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero, "A uniform solution to SAT using membrane creation," *Theoretical Computer Science*, vol. 371, no. 1-2, pp. 54–61, 2007.
[4] V. Manca, "DNA and membrane algorithms for SAT," *Fundamenta Informaticae*, vol. 49, no. 1-3, pp. 205–221, 2002.
[5] L. Q. Pan and A. Alhazov, "Solving HPP and SAT by P systems with active membranes and separation rules," *Acta Informatica*, vol. 43, no. 2, pp. 131–145, 2006.
[6] G. Păun, "P systems with active membranes: Attacking NP-complete problems," *Journal of Automata, Languages and Combinatorics*, vol. 6, no. 1, pp. 75–90, 2001.
[7] M. J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini, "A polynomial complexity class in P systems using membrane division," *Journal of Automata, Languages and Combinatorics*, vol. 11, no. 4, pp. 423–434, 2003.
[8] C. Zandron, C. Ferretti, and G. Mauri, "Solving NP-complete problems using P systems with active membranes," in *Unconventional Models of Computation*, 2000, pp. 289–301.
[9] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez, "A fast P system for finding a balanced 2-partition," *Soft Computing*, vol. 9, no. 9, pp. 673–678, 2005.

[10] M. J. Pérez-Jiménez and A. Riscos-Núñez, "A linear-time solution to the knapsack problem using P systems with active membranes," *Membrane Computing*, vol. 2933, pp. 250–268, 2004.
[11] ——, "Solving the subset-sum problem by P systems with active membranes," *New Generation Computing*, vol. 23, no. 4, pp. 339–356, 2005.
[12] M. J. Pérez-Jiménez and F. Romero-Campero, "Solving the BIN PACKING problem by recognizer P systems with active membranes," in *The Second Brainstorming Week on Membrane Computing*, 2004, pp. 414–430.
[13] A. Leporati, C. Zandron, and G. Mauri, "Solving the factorization problem with P systems," *Progress in Natural Science*, vol. 17, no. 4, pp. 471–478, 2007.
[14] A. Fujiwara and T. Tateishi, "Logic and arithmetic operations with a constant number of steps in membrane computing," *International Journal of Foundations of Computer Science*, vol. 22, no. 3, pp. 547–564, 2011.
[15] R. Freund, "Asynchronous P systems and P systems working in the sequential mode," in *International workshop on Membrane Computing*, 2005, pp. 36–62.
[16] T. Murakawa and A. Fujiwara, "Arithmetic operations and factorization using asynchronous P systems," *International Journal of Networking and Computing*, vol. 2, no. 2, pp. 217–233, 2012.
[17] 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)*, vol. E95-D, no. 3, 2012.
[18] K. Tanaka and A. Fujiwara, "Asynchronous P systems for hard graph problems," *International Journal of Networking and Computing*, vol. 4, no. 1, pp. 2–22, 2014.
[19] N. Haberman, "Parallel neighbor-sort (or the glory of the induction principle)," AD-759 248, National Technical Information Service, US Department of Commerce, Tech. Rep., 1972.