Abstract—In the present paper, we consider the reaction system, which is a computational model based on biochemical reactions in living cells, and propose reaction systems for logical operations and sorting. We first propose a reaction system that executes a two-input logical operation, such as AND, OR, and XOR, and show that the reaction system works in $O(1)$ steps. We next propose a reaction system for a compare-and-swap operation of two binary numbers of $m$ bits. We show that the reaction system works in $O(m)$ parallel steps using $O(m)$ types of objects and reaction rules. We finally propose a reaction system for sorting of $n$ binary numbers of $m$ bits. The reaction system is based on an idea of the odd-even sort, and we show that the reaction system works in $O(mn)$ parallel steps and using $O(mn)$ types of objects and reaction rules.

I. INTRODUCTION

A number of next-generation computing paradigms have been considered due to limitation of silicon based computation. As an example of the computing paradigms, natural computing, which works using natural materials for computation, has considerable attention. A membrane computing [1], which is a computational model inspired by the structures and behaviors of living cells, is a representative of the natural computing. A computational model of the membrane computing is called P system, and a number of P systems [2], [3], [4], [5], [6], [7] have been proposed for solving NP problems. In addition, a number of P systems [2], [8] are also proposed for basic operations such as logical and arithmetic operations.

On the other hand, a reaction system [9], [10], [11], which is called R system, has been proposed as another computational model of natural computing. The R system is based on biochemical reactions in living cells, and the fundamental idea of the R system is based on interaction between biochemical reactions, which are the mechanisms of facilitation and inhibition. For the reaction system, a number of primitive operations are considered in [9], [10], [11], and no R system that executes basic operations, such as logic or arithmetic operations, has been proposed. However, the reaction system for the basic operation is needed to apply the reaction system on a wide range of problems.

In the present paper, we propose R systems for logical operations and sorting of binary numbers. We first propose a reaction system that executes a two-input logical operation, such as AND, OR, and XOR. We show that the R system works in $O(1)$ parallel steps and using $O(1)$ types of objects and reaction rules.

We next propose an R system for a compare-and-swap operation of two binary numbers of $m$ bits. The R system first computes the most significant bit between the two input values, and then, swap operation is executed according to the result of the most significant bit. We show that the R system works in $O(m)$ parallel steps and using $O(m)$ types of objects and reaction rules.

We finally propose an R system for sorting of $n$ binary numbers of $m$ bits. The R system is based on an idea of the odd-even sort, and the R system employs an object that works as a counter, and executes the sorting for odd and even steps using the counter. We show that the R system works in $O(mn)$ parallel steps and using $O(mn)$ types of objects and reaction rules.

II. PRELIMINARIES

A. Reaction system

A reaction system [9], [10], [11] is a computational model based on biochemical reactions in living cells. In this paper, we first explain definition of a reaction on the reaction system, which is based on [11].

A reaction $a$ is defined by the following equation.

$$a = (R_a, I_a, P_a)$$

$R_a, I_a$ and $P_a$ are sets of reactant, inhibitor and product, respectively, and all of the three sets are finite nonempty sets such that $R_a \cap I_a = \emptyset$, $M = R_a \cup I_a$, and $|M| \geq 2$.

The reaction $a$ is applied if $R_a \subseteq T$ and $I_a \cap T = \emptyset$ for a finite set $T$. The result of $a$ on $T$ is denoted by $Res_a(T)$, and $Res_a(T) = P_a$ in case that reaction $a$ is applied, and otherwise, $Res_a(T) = \emptyset$.

I now show an example of the reaction. Let $a = (\{3\}, \{1, 2\}, \{1, 2, 4\})$ and $T_1 = \{3, 4\}$, $T_2 = \{2, 3, 4\}$. In this case, $Res_a(T_1)$ and $Res_a(T_2)$ are sets given below.

$$Res_a(T_1) = P_a = \{1, 2, 4\}$$

$$Res_a(T_2) = \emptyset$$

As shown in the above example, all non-reacted objects, which are not included in $P_a$, are disappeared after application.
of reaction \(a\). The property is called non-persistency of the object.

Next, we explain definition of the R system. R system \(\Lambda\) is defined by the following equation.

\[
\Lambda = (S, A)
\]

In the above equation, \(S\) is a set of all objects, and \(A\) is a set of reactions. In addition, the result of a set of reactions \(A\) for \(T\) is an union of obtained results of all reactions in \(A\). In other words, \(Res_A(T)\), which is a set of result of \(A\) for a finite set \(T\), is defined as follows.

\[
Res_A(T) = \bigcup_{a \in A} res_a(T)
\]

In the R system, an application of all reaction is called a transition. In this paper, we assume that all applicable reactions are applied simultaneously in a transition, and also assume that each transition is called one parallel step. The complexity of R system is defined the number of parallel steps executed in the computation.

For example, we show a simple R system \(\Lambda\), which is defined as follows.

\[
\begin{align*}
S &= \{1, 2, 3, 4\} \\
A &= \{a, b, c\} \\
a &= \{(3), \{1, 2\}\} \\
b &= \{(3, 4), \{2\}, \{4\}\} \\
c &= \{(1, 4), \{3\}, \{2, 3\}\}
\end{align*}
\]

Figure 1 shows an execution of the R system in case that a set \(\{3, 4\}\) is given as input. In this case, applicable reactions are \(a\) and \(b\) for the input set, and these two rules are applied simultaneously. Since a result of the transition is an union of obtained objects, an obtained set of objects is \(\{1, 2, 4\}\) after the first transition.

Next, an applicable reaction for \(\{1, 2, 4\}\) is only \(c\), and an obtained set of objects is \(\{2, 3\}\) after the second transition. Then, a set of objects becomes empty after the third transition since no reaction is applicable for \(\{2, 3\}\).

III. DATA STRUCTURE FOR BINARY NUMBERS

In this subsection, we describe a unified data structure for a binary number using objects in the R system. Data structure for Boolean values has been proposed in [2], and we improve the data structure in this paper. In the data structure, 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 data structure enables the addressing feature, that is, each binary number is stored in a given address.

Let \(V_{i,j} \in \{0, 1\}\) be a \(j\)-th Boolean value stored in address \(i\). Then, the value is denoted using the following object on the R system.

\[
\langle A_i, B_j, V_{i,j} \rangle
\]

We call the above object a memory object for Boolean values.

In case of a binary number, let \(V_{i,m-1}, V_{i,m-2}, \cdots, V_{i,0}\) be \(m\) Boolean values stored in address \(i\). Then, a non-negative integer \(V_i\) stored in address \(i\) satisfies the following condition.

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

The above binary number is represented by \(m\) memory objects given below.

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

For example, the binary number 1010 stored in the address 6 is represented by the following four objects.

\[
\langle A_6, B_3, 1 \rangle, \langle A_6, B_2, 0 \rangle, \langle A_6, B_1, 1 \rangle, \langle A_6, B_0, 0 \rangle
\]

IV. R SYSTEMS FOR LOGICAL OPERATIONS

A. Input and output

In this section, we propose a simple R system that executes a two-input logical operation, such as AND, OR, and XOR. We assume that input of the logical operation is a pair of Boolean values \(x, y\). The \(x, y\) are denoted by a pair of following two memory objects on the R system.

\[
\langle A_0, B_0, V_{0,0} \rangle, \langle A_1, B_0, V_{1,0} \rangle
\]

We also assume that an output of the logical operation is a Boolean value \(z\), which is denoted by the following memory object.

\[
\langle A_2, B_0, V_{2,0} \rangle
\]

B. R system for logical operations

In this subsection, we show an R system for the logical operation. Any two-input logical operation is defined in the truth table in Table I. For example, \(z_0 = z_1 = z_2 = 0\) and \(z_3 = 1\) in case of AND operation. We propose an R system for any two-input logical operation based on the truth table.

C. Details of the R system for the logical operation

An output of the logical operation is determined according to the truth table. Thus, we encode the truth table into the R system as reactions. We now formally define the R system \(\Lambda_{LO}\) for any logical operations in the following.

\[
\Lambda_{LO} = (S, A)
\]
TABLE I
A TRUTH TABLE OF A TWO-INPUT LOGICAL OPERATION

<table>
<thead>
<tr>
<th>Input x</th>
<th>Input y</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$z_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$z_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$z_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$z_3$</td>
</tr>
</tbody>
</table>

Let $x_{in}, y_{in}$ be $j$-th bit of two input binary numbers.

B. An overview of the R system

Let $x_{in,j}$ and $y_{in,j}$ be $j$-th bit of two input binary numbers.
The R system for the compare-and-swap consists of the following two steps. First, in Step 1, each bit of the two input binary numbers, $x_{in}$ and $y_{in}$, are compared from a higher bit to a lower bit. We assume that $x_{in,k}$ and $y_{in,k}$ are the first pair of bits such that the bits are different in the comparison. Then, $x_{in}$ is greater than $y_{in}$ if $x_{in,k} = 1, y_{in,k} = 0$, otherwise, $x_{in}$ is less than $y_{in}$, and $x_{in,k} = 0, y_{in,k} = 1$. An object that denotes a result of the comparison is created after the comparison.

Second, in Step 2, two binary numbers are exchanged and outputted to $x_{out}$ and $y_{out}$ in case of $x_{in} < y_{in}$. Otherwise, $x_{in}$ and $y_{in}$ are copied into $x_{out}$ and $y_{out}$, respectively.

In the above two steps, it is worth while noticing that memory objects must be copied repeatedly because of non-persistency of the object.

C. Details of the R system

We now explain each step of the R system. In Step 1, each bit of the two input binary numbers, $x_{in}$ and $y_{in}$, are compared from a higher bit to a lower bit. The comparison is executed using the following sets of reactions.

$$A_{1,1} = \{(\{A_0, B_k, 0\}, \{A_1, B_k, 1\}, \{E\}), \{\{A_0, B_k, 0\}, \{A_1, B_k, 1\}, \{LT\}\} \mid 0 \leq k \leq m - 1\}$$

$$A_{1,2} = \{(\{A_0, B_k, 0\}, \{A_1, B_k, 1\}, \{C, k\}\} \mid 1 \leq k \leq m - 1\}$$

In case of $j$-th bit of two binary numbers are different, reactions in $A_{1,1}$ are applied, and one of objects, $\langle LT \rangle$ or $\langle GT \rangle$, which denotes $x_{in} < y_{in}$ or $x_{in} > y_{in}$, is created. Otherwise, two bits are copied, and the comparison is moved to the next bit-position using reactions in $A_{1,2}$.

In addition to the above, the other memory objects are copied repeatedly, due to non-persistency of the object, using the following sets of reactions.

$$A_{1,3} = \{(\{A_0, B_k, V_{0,k}\}, \{C, k\}, \{LT\}, \{GT\}, \{EQ\}\}, \{\{A_0, B_k, V_{0,k}\} \mid 0 \leq k \leq m - 1, V_{0,k} \in \{0, 1\}\}$$

$$\cup \{(\{A_1, B_k, V_{1,k}\}, \{C, k\}, \{LT\}, \{GT\}, \{EQ\}\}, \{\{A_1, B_k, V_{1,k}\} \mid 0 \leq k \leq m - 1, V_{1,k} \in \{0, 1\}\}$$

In case of $x_{in} = y_{in}$, reactions in the following $A_{1,4}$ is applied after comparisons of all bits, and an object $\langle EQ \rangle$ is
A. Complexity of the R system

Since complexity of Step 1 in the above R system $\Lambda_{CS}$ is $O(m)$, we obtain the following theorem for $\Lambda_{CS}$.

**Theorem 2:** The R system $\Lambda_{CS}$, which executes the compare-and-swap operation for two binary numbers of $m$ bits, works in $O(m)$ parallel steps using $O(m)$ types of objects and reactions.

VI. SORTING

A. Input and output

In this section, we present an R system for sorting of $n$ binary numbers of $m$ bits using the R system. An input and an output of the R system is a set of binary numbers that are denoted by the following set of memory objects.

$$\{(A_i, B_j, V_{i,j}) | 0 \leq i \leq n-1, 0 \leq j \leq m-1\}$$

B. An overview and complexity of the R system

The proposed P system is based on odd-even transposition sort [12], which is a well-known parallel sorting algorithm. Let
(x_0, x_1, \ldots, x_{n-1})$ be an input of the sorting. A basic idea of the odd-even transposition sort is quite simple. At odd phases, we perform the compare-and-swap operations for each pair $(x_{2i}, x_{2i+1})$ $(0 \leq i \leq \frac{n}{2} - 1)$ in parallel. On the other hand, we perform the same operations for each pair $(x_{2i-1}, x_{2i})$ $(1 \leq i \leq \frac{n}{2} - 1)$ at even phases. It is proved that the input is sorted after $\frac{n}{2}$ repetition of the above two steps \[12\].

Using the R system described in Section V, we can realize the odd-even transposition sort on the R system as follows.

**A basic idea of R system for sorting**

Repeat the following steps $\frac{n}{2}$ times.

1. **Step 1:** Execute the compare-and-swap operations, which is described in Section V, for pairs $(x_{2i}, x_{2i+1})$ $(0 \leq i \leq \frac{n}{2} - 1)$ in parallel.

2. **Step 2:** Execute the same compare-and-swap operations for pairs $(x_{2i-1}, x_{2i})$ $(1 \leq i \leq \frac{n}{2} - 1)$ in parallel.

Since all reactions on the R system can be applied in parallel, the above idea is implemented as an R system $\text{SORT}$, which sorts $n$ binary number of $m$ bits, with a modification of the R system proposed in Section V. Although the main difficulty for the implementation is synchronization of the compare-and-swap operations, we can solve the difficulty by using object that works as a global counter. (The precise description of the R system is omitted because an implementation of reactions is redundant.)

We now consider complexity of the R system $\text{SORT}$. The above two steps are repeated by $\frac{n}{2}$ times, and each compare-and-swap operation is executed in $O(m)$ steps. Then, we obtain the following theorem for the R system $\text{SORT}$.

**Theorem 3:** The R system $\text{SORT}$, which sorts $n$ binary numbers of $m$ bits, works in $O(mn)$ parallel steps using $O(mn)$ types of objects and reactions.

**VII. Conclusions**

In the present paper, we proposed R systems for logical operations and sorting of binary numbers. We first proposed an R system that executes any two-input logical operation, and showed that the R system works in $O(1)$ parallel steps and using $O(1)$ types of objects and reaction rules. We next proposed an R system for a compare-and-swap operation of two binary numbers of $m$ bits, and showed that the R system works in $O(m)$ parallel steps and using $O(m)$ types of objects and reaction rules. We finally proposed an R system for sorting of $n$ binary numbers of $m$ bits, and also showed that the R system works in $O(mn)$ parallel steps and using $O(mn)$ types of objects and reaction rules.

As future work, we are considering reduction of the numbers of types of objects and reactions in the R systems.

**Acknowledgments**

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