International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 2, pages 217–233, July 2012

Arithmetic Operations and Factorization using Asynchronous P Systems

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

Received: January 23, 2012
Revised: May 17, 2012
Accepted: June 19, 2012
Communicated by Sayaka Kamei

Abstract

In the present paper, we consider the asynchronous parallelism in membrane computing, and propose asynchronous P systems that perform two basic arithmetic operations and factorization. Since there is no restrictive assumption for application of rules, sequential and maximal parallel executions are allowed on the asynchronous P system.

We first propose a P system that computes *addition* of two binary numbers of m bits. The P system works in $O(m)$ sequential and parallel steps using $O(m)$ types of objects. We next propose a P system for *multiplication* of the two binary numbers of m bits, and show that the P system works in $O(m \log m)$ parallel steps or $O(m^3)$ sequential steps using $O(m^2)$ types of objects. Finally, we propose a P system for *factorization* of a positive integer of m bits using the above P system as a sub-system. The P system computes the factorization in $O(m \log m)$ parallel steps or $O(4^m \times m^2 \log m)$ sequential steps using $O(m^2)$ types of objects.

Keywords: membrane computing, P system, addition, multiplication, factorization

1 Introduction

Membrane computing, which is a representative example of natural computing, is a computational model inspired by the structures and behaviors of living cells. In the initial study on membrane computing, a basic feature of the membrane computing was introduced by Păun et al. [13] as P system. Each P system consists of hierarchically embedded cell membranes, and each membrane may contain objects. Each object evolves according to evolution rules associated with a membrane in which the object is contained. Several P systems have been proposed for numerical NP problems [9, 14, 15, 16] using the maximal parallelism, which is a main feature of the P system.

However, the P systems for primitive operations, such as logic or arithmetic operations, are needed to apply the membrane computing on a wide range of problems. A number of P systems [8, 10, 11, 12, 18] have been proposed for basic arithmetic operations in membrane computing. Leporati et al. [12] proposed three P systems for arithmetic operations. The first P system computes addition of two binary numbers of m bits in $O(m)$ steps, and the second P system computes multiplication of two binary numbers of m bits in $O(m \log m)$ steps. The two P systems are used in the third P system, which computes factorization of a natural number of m bits in $O(m \log m)$ steps, as sub-systems. For another example, Fujiwara et al. [8] proposed a P system that computes addition of two binary numbers of m bits in a constant number of steps. In these P systems, synchronous application of evolution rules is assumed with the maximal parallelism.

However, 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 the P system a more realistic model.

For considering the asynchronous parallelism, a number of P systems [1, 2, 3, 5, 6, 7] have been proposed. For example, some sequential P systems [5, 7], which assume sequential application of applicable evolution rules, have been proposed. In the papers, the computational powers of the sequential P systems are considered, and the P systems has been proved to be universal. For another example, P systems with minimal parallelism are considered in [3] The P system is proved to be universal, and a P system has been proposed for solving SAT in $O(n)$ steps. However, on the P system with minimal parallelism, at least one rule is applied in each membrane if there exists a rule that can be applied for the membrane. In other words, applicable rules in another membranes are applied synchronously, and some kind of synchronicity is assumed in the model. In addition, time-free P systems are considered in [1, 2]. Although the time-free P systems has also been proved to be universal, no P system has been proposed for solving a hard problem.

Recently, two P systems for NP problems are proposed with asynchronous parallelism in [17]. The first and second P systems solve SAT and Hamiltonian problems in polynomial numbers of steps, respectively. However, no known asynchronous P systems have been proposed for the basic arithmetic operations, and we propose P systems for computing the operations using the asynchronous parallelism.

In the present paper, we consider fully asynchronous parallelism such that any number of applicable rules may be applied in one step on the asynchronous P system. Since there is no restrictive assumption for application of rules, sequential and fully parallel executions are allowed on the asynchronous P system. As complexity of the asynchronous P system, we consider two kinds of numbers, which are the number of sequential steps and the number of parallel steps. The numbers of sequential steps is the number of executed steps in case that rules are applied sequentially, and the number of parallel steps is the number of executed steps with maximal parallelism.

Using the asynchronous parallelism and the objects with the addressing feature, we propose three P systems that perform two basic arithmetic operations and factorization. In the three P systems, each number is denoted as a binary number.

We first propose an asynchronous P system that computes *addition* of two binary number of m bits. The P system works in $O(m)$ steps using $O(m)$ types of objects in both of maximally parallel and sequential manners. We next propose an asynchronous P system for *multiplication* of two binary numbers of m bits using the above P system as a sub-system. The P system works in $O(m \log m)$ parallel steps or $O(m^3)$ sequential steps using $O(m^2)$ types of objects.

Finally, we propose an asynchronous P system for factorization. The input of the P system is a positive integer of m bits, which has two prime factors, and the output of the P system is a pair of the two prime factors. The P system for multiplication is used as a sub-system in the P system, and the number of the sub-systems is exponentially increased using division of membranes in a polynomial number of steps. The proposed P system computes the factorization in $O(m \log m)$ parallel steps or $O(4^m \times m^2 \log m)$ sequential steps using $O(m^2)$ types of objects.

Although the numbers of parallel steps of the proposed P systems are not smaller than synchronous P systems [8, 12] for the same operations, the proposed P systems work under any kinds of assumptions for synchronicity. In other words, the proposed P systems work on all of synchronous, sequential and fully asynchronous P systems.

The paper is organized as follows. In Section 2, we give a brief description of the model for membrane computing and show representation of binary numbers. In Section 3 and Section 4, we propose P systems for addition and multiplication, respectively. We also propose a P system for factorization in Section 5. Finally, we conclude the paper in Section 6.

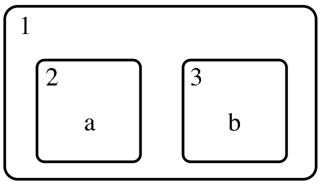

Figure 1: An example of membrane structure

2 Preliminaries

2.1 Computational model for membrane computing

Several models [4] have been proposed for membrane computing. We briefly introduce a basic model of the P system in this subsection.

The P system consists of membranes and objects. Membranes are labeled using distinct symbols and form nested structures. In the present paper, each membrane is denoted 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. In addition, each membrane can be polarized with $+$, $-$ or 0.

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

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

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

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

H: H is a set of labels for membranes. We assume that a membrane labeled *skin*, which is called the skin membrane, is the outermost membrane, i.e., the skin membrane contains all of the other membranes.

μ: μ is membrane structure that consists of m membranes. Each membrane in the structure is labeled with an element in H. (For example, μ in the above example is $[\ [\ ]_2\ [\ ]_3\ ]_1$.)

$\omega_1, \omega_2, \cdots, \omega_m$: Each ω_i is a set of objects initially contained only in the membrane labeled i. (In the above example, $\omega_1 = \phi$, $\omega_2 = \{a\}$, and $\omega_3 = \{b\}$.)

$R_1, R_2, \cdots, R_m$: Each R_i is a set of evolution rules that are applicable to objects in the i-th membrane.

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:

$$[\ \alpha\ ]_h^{e_1} \rightarrow [\ \beta\ ]_h^{e_2}$$

In the rule, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, and $\alpha, \beta \in O$. Using the rule, an object α evolves into another object, β. In addition, the membrane can be polarized with $+$, $-$, or 0. (For the case in which the value of the polarization is 0, the label is omitted.)

(2) Send-in communication rule:

$$\alpha\ [\ ]_h^{e_1} \rightarrow [\ \beta\ ]_h^{e_2}$$

In the rule, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, and $\alpha, \beta \in O$. Using the rule, an object α is sent in the membrane, and can evolve into another object β. The membrane can also be polarized.

(3) Send-out communication rule:

$$[\ \alpha\ ]_h^{e_1} \rightarrow [\ ]_h^{e_2}\ \beta$$

In the rule, $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, and $\alpha, \beta \in O$. Using the rule, an object α is sent out of the membrane, and can evolve into another object β. The membrane can also be polarized.

(4) Dissolution rule:

$$[\ \alpha\ ]_h^{e_1} \rightarrow \beta$$

In the rule, $h \in H$, $e_1 \in \{+, -, 0\}$, and $\alpha, \beta \in O$. Using the rule, the membrane, which contains object α, is dissolved, and the object can evolve into another object β. (The skin membrane cannot be dissolved.)

(5) Division rule:

$$[\ \alpha\ ]_h^{e_1} \rightarrow [\ \beta\ ]_h^{e_2}\ [\ \gamma\ ]_h^{e_3}$$

In the rule, $h \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$, and $\alpha, \beta, \gamma \in O$. Using the rule, the membrane, which contains object α, is divided into two membranes that contain objects β and γ, respectively. The membranes can also be polarized.

We omit the brackets in each evolution rule for cases in which the membrane, to which the evolution rule is applied, is obvious in the following description.

We also assume that each of the above rules is applied in a constant number of biological steps, and consider the number of steps executed in P systems as the complexity of the P systems.

2.2 Maximal parallelism and asynchronous parallelism

All of the above evolution rules are applied in a non-deterministic maximally parallel manner on the standard P system. In one step of the standard 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 objects, 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 assume that evolution rules are applied in fully asynchronous manner as in [8]. On the asynchronous P system, at least one of applicable evolution rules must be applied in each step of computation. In other words, the asynchronous P system can be executed sequentially, and can also be executed in maximally parallel manner. We call the number of steps executed on the asynchronous P system in maximally parallel manner *the number of parallel steps*. On the other hand, various sequential executions can be considered on the asynchronous P system. We call the worst number of sequentially executed steps *the number of sequential steps*.

For the asynchronous P system, the number of parallel and sequential steps means the best and worst case complexities, respectively. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

We now show an example for difference between the P system with maximal parallelism and the asynchronous P system. The following Π_1 is a simple P system that computes AND function for any number of Boolean values.

$$\Pi_1 = (O, H, \mu, \omega_1, R_1),$$

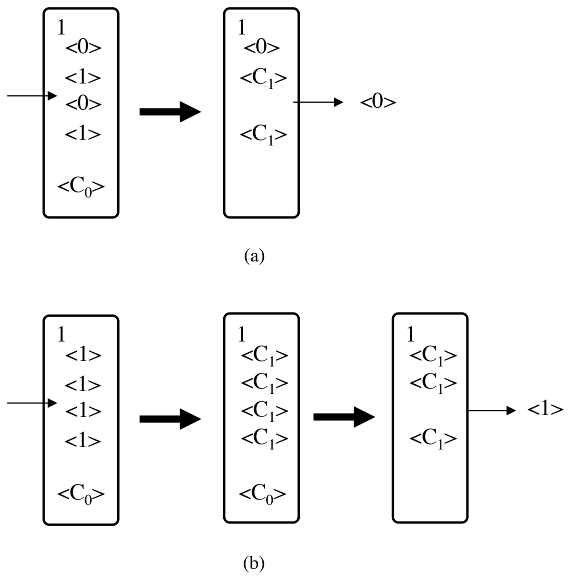

Figure 2: An example of executions of the P system Π_1 with maximal parallelism: (a) an execution in case that there exists $\langle 0\rangle$ in the input, (b) an execution in cast that all input objects are $\langle 1\rangle$.

- $O = \{\langle 0\rangle, \langle 1\rangle, \langle C_0\rangle, \langle C_1\rangle\}$
- $H = \{1\}$
- $\mu = [\ ]_1$
- $\omega_1 = \{\langle C_0\rangle\}$
- $R_1 = R_{1,1} \cup R_{1,2}$
  - $R_{1,1} = \{[\langle 0\rangle\langle C_0\rangle] \rightarrow [\ ]\langle 0\rangle,\ \langle 1\rangle \rightarrow \langle C_1\rangle\}$
  - $R_{1,2} = \{\langle C_0\rangle\langle C_1\rangle \rightarrow [\ ]\langle 1\rangle\}$

In the above P system, two objects, $\langle 0\rangle$ and $\langle 1\rangle$, denote Boolean values 0 and 1, respectively.

Figure 2 (a) illustrates an execution in case that there exists object $\langle 0\rangle$ in the input set, and Figure 2 (b) illustrates an execution in case that there exists no $\langle 0\rangle$ in the input set, i.e. all input objects are $\langle 1\rangle$. In both cases, the P system Π_1 computes AND function for any number of Boolean values in a constant number of steps with maximal parallelism.

However, P system Π_1 may output wrong value $\langle 1\rangle$ in case that both objects, $\langle 0\rangle$ and $\langle 1\rangle$, are in the input set on any kinds of asynchronous P systems.

In fact, computation of AND function for any number of Boolean values is a hard problem on asynchronous P systems. The following Π_2 is a P system that asynchronously computes AND function for n Boolean values for a given n.

$$\Pi_2 = (O, H, \mu, \omega_1, R_1),$$

- $O = \{\langle 0\rangle, \langle 1\rangle, \langle C_0\rangle, \langle C_1\rangle, \cdots \langle C_n\rangle\}$
- $H = \{1\}$
- $\mu = [\ ]_1$
- $\omega_1 = \{\langle C_0\rangle\}$
- $R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$

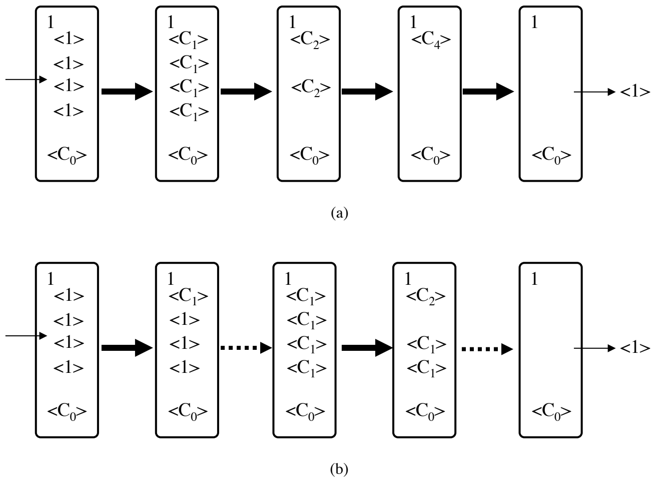

Figure 3: An example of executions of the P system Π_2: (a) an execution with maximal parallelism, (b) a sequential execution.

- $R_{1,1} = \{[\langle 0\rangle\langle C_0\rangle] \rightarrow [\ ]\langle 0\rangle, \langle 1\rangle \rightarrow \langle C_1\rangle\}$
- $R_{1,2} = \{\langle C_i\rangle\langle C_j\rangle \rightarrow \langle C_{i+j}\rangle \mid 1 \leq i \leq j \leq n-1\}$
- $R_{1,3} = \{\langle C_n\rangle \rightarrow [\ ]\langle 1\rangle\}$

We now describe complexity of asynchronous P systems using the above example. We first introduce complexity of an asynchronous P system in maximal parallel manner. Since P system Π_2 can be obviously executed in one step if $\langle 0\rangle$ is in the input set, we consider the case that P system Π_2 is executed for an input set that includes no $\langle 0\rangle$.

Figure 3 (a) illustrates an execution in case that P system Π_2 is executed in maximal parallel manner for the input set. In the execution, object C_n is created in $O(\log n)$ steps since the number of objects C_i is reduced by half in each step of the execution. We call the number of steps executed on the asynchronous P system in maximal parallel manner *the number of parallel steps*. For example, P system Π_2 computes the AND function in $O(\log n)$ parallel steps.

We next introduce complexity of an asynchronous P system in case that the P system is executed sequentially. Figure 3 (b) illustrates an example of execution in case that P system Π_2 is executed sequentially for the same input set. The computation is executed in $O(n)$ steps if one evolution rule is applied in each step of the execution. Since various sequential executions can be considered on the asynchronous P system, we call the worst number of sequentially executed steps *the number of sequential steps*. For example, P system Π_2 computes the AND function in $O(n)$ sequential steps.

For the asynchronous P system, the number of parallel and sequential steps means the best and worst case complexities, respectively. In addition to this, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

2.3 Representation of binary numbers with objects

In this subsection, we describe a unified representation of a binary number with objects. The representation is the same as in [8], 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.

Let $V_{i,m-1}, V_{i,m-2}, \cdots, V_{i,0}$ be m Boolean values stored in address i. In case 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 the above value. 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*.

3 Addition

3.1 Input and output

We assume that V_x and V_y are two input numbers, such that $V_x = \sum_{j=0}^{m-1} V_{x,j} \times 2^j$ and $V_y = \sum_{j=0}^{m-1} V_{y,j} \times 2^j$. The two numbers V_x and V_y are stored in the following two sets, O_x and O_y.

$$\begin{aligned} O_x &= \{\langle A_x, B_j, V_{x,j}\rangle \mid 0 \le j \le m-1\} \\ O_y &= \{\langle A_y, B_j, V_{y,j}\rangle \mid 0 \le j \le m-1\} \end{aligned}$$

We assume that the above two sets of memory objects are given from the outside region into the skin membrane.

The output of the addition is stored in the following set O_s, i.e., $V_s = V_x + V_y$ such that $V_s = \sum_{j=0}^{m} V_{s,j} \times 2^j$.

$$O_s = \{\langle A_s, B_j, V_{s,j}\rangle \mid 0 \le j \le m\}$$

We also assume that the above set of objects is sent from the skin membrane to the outside region.

For example, the following two sets, O_x and O_y, denote two input binary numbers, $V_x = (1011)_2$ and $V_y = (1101)_2$.

$$\begin{aligned} O_x &= \{\langle A_x, B_0, 1\rangle, \langle A_x, B_1, 1\rangle, \langle A_x, B_2, 0\rangle, \langle A_x, B_3, 1\rangle\} \\ O_y &= \{\langle A_y, B_0, 1\rangle, \langle A_y, B_1, 0\rangle, \langle A_y, B_2, 1\rangle, \langle A_y, B_3, 1\rangle\} \end{aligned}$$

In addition, the following set O_s denotes an output $V_s = (11000)_2$, which is the result of addition of the above two numbers.

$$O_s = \{\langle A_s, B_0, 0\rangle, \langle A_s, B_1, 0\rangle, \langle A_s, B_2, 0\rangle, \langle A_s, B_3, 1\rangle, \langle A_s, B_4, 1\rangle\}$$

3.2 An asynchronous P system for addition

We first explain an overview of the asynchronous P system for computing addition. In [8], a standard P system, which computes addition for the same input, has been proposed. Since values of all bits can be synchronously computed on the standard P system, the P system computes the addition in a constant number of steps. However, we cannot compute addition for all bits in parallel on the asynchronous P system, and the addition is sequentially computed from the lowest bit to higher bits.

The computation of the asynchronous P system is executed in 3 steps given below.

Step 1: Compute $V_{s,0} = V_{x,0} \oplus V_{y,0}$ and $V_{c,1} = V_{x,0} \wedge V_{y,0}$.
(In the computation, $V_{s,0}$ denotes a value of addition in the lowest bit, and $V_{c,0}$ denotes a value of a carry to the next bit.)

Step 2: Compute $V_{s,j}$ and $V_{c,j+1}$ from $j = 1$ to $m-1$ as follows.

- $V_{s,j} = V_{x,j} \oplus V_{y,j} \oplus V_{c,j}$.
- $V_{c,j+1}$ is 1 if at least two bits of $V_{x,j}$, $V_{y,j}$ and $V_{c,j}$ are 1. Otherwise $V_{c,j+1} = 0$.

Then, $V_{s,m} = V_{c,m}$.

Table 1: A truth table for R_2.

$V_{x,j}$	$V_{y,j}$	$V_{c,j}$	$V_{s,j}$	$V_{c,j+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 3: Output memory objects that denote V_s to the environment.

We now explain each set of evolution rules that realize the above 3 steps. The membrane structure used in the computation is skin membrane only. Step 1 is executed applying the following set R_1 for input objects.

$$\begin{aligned} R_1 \;=\; & \{\langle A_x, B_0, 0\rangle\langle A_y, B_0, 0\rangle \to \langle A_s, B_0, 0\rangle\langle A_c, B_1, 0\rangle, \\ & \langle A_x, B_0, 0\rangle\langle A_y, B_0, 1\rangle \to \langle A_s, B_0, 1\rangle\langle A_c, B_1, 0\rangle, \\ & \langle A_x, B_0, 1\rangle\langle A_y, B_0, 0\rangle \to \langle A_s, B_0, 1\rangle\langle A_c, B_1, 0\rangle, \\ & \langle A_x, B_0, 1\rangle\langle A_y, B_0, 1\rangle \to \langle A_s, B_0, 0\rangle\langle A_c, B_1, 1\rangle\} \end{aligned}$$

In Step 2, addition of each bit is executed applying the following set of evolution rules, R_2. (Boolean values in R_2 are defined in a truth table in Table 1.)

$$\begin{aligned} R_2 \;=\; & \{\langle A_x, B_j, V_{x,j}\rangle\langle A_y, B_j, V_{y,j}\rangle\langle A_c, B_j, V_{c,j}\rangle \to \langle A_s, B_j, V_{s,j}\rangle\langle A_c, B_{j+1}, V_{c,j+1}\rangle \\ & \mid 1 \le j \le m-1\} \\ & \cup\{\langle A_c, B_m, V\rangle \to \langle A_s, B_m, V\rangle \mid V \in \{0,1\}\} \end{aligned}$$

In Step 3, output memory objects are sent out from the skin membrane applying the following set of send-out communication rules, R_3.

$$R_3 = \{[\langle A_s, B_j, V\rangle] \to [\;]\langle A_s, B_j, V\rangle \mid 0 \le j \le m, V \in \{0,1\}\}$$

We now summarize P system Π_{ADD} that asynchronously computes addition of two binary numbers of m bits.

$$\Pi_{ADD} = (O, \mu, \omega, R),$$

- $O = O_x \cup O_y \cup O_s \cup \{\langle A_c, B_j, V\rangle \mid 0 \le j \le m, V \in \{0,1\}\}$
- $\mu = [\;\;]_1$
- $\omega = O_x \cup O_y$
- $R = R_1 \cup R_2 \cup R_3$

Figure 4 illustrates an execution of the P system Π_{ADD}. In this example, two input numbers are 1011 and 1101. Two set of objects that denote the input numbers are given from the outside region into the skin membrane. Then, sets of evolution rules, R_1 and R_2, are applied, and the addition is computed. Finally the result is sent out from the membrane using R_3.

It is worth while noticing that there are various executions for Π_{ADD} since the P system is fully asynchronous. For example, an output object $\langle A_s, B_0, 0\rangle$ may be sent out from the membrane before the other output object $\langle A_s, B_3, 1\rangle$ is created. In any asynchronous execution, the proposed P system Π_{ADD} outputs a correct result for addition of two binary numbers.

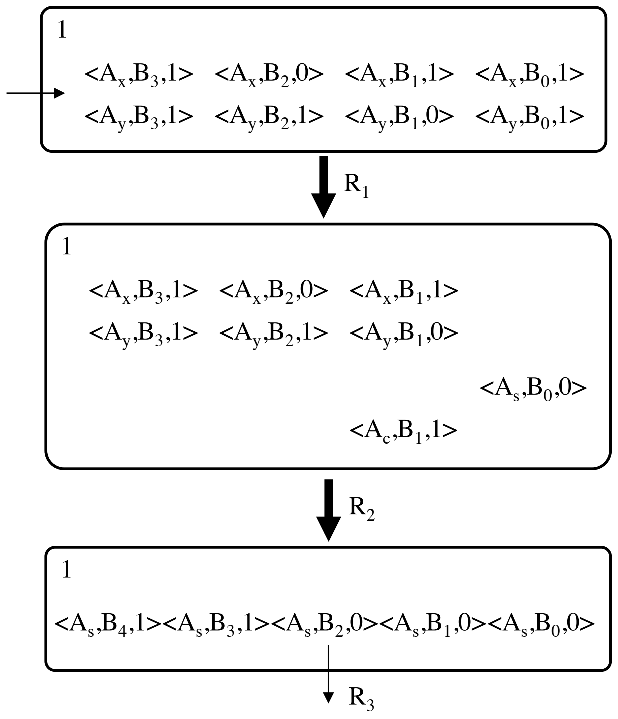

Figure 4: An example of execution of Π_{ADD}.

3.3 Complexity

We consider complexity of the proposed P system Π_{ADD}. Both of sequential and parallel steps are $O(m)$, and the number of objects and the number of evolution rules used in the system are $O(m^2)$. Therefore, we obtain the following theorem for Π_{ADD}.

Theorem 1 *The P system Π_{ADD}, which computes addition of two binary numbers of m bits, works in $O(m)$ parallel steps or $O(m)$ sequential steps using $O(m^2)$ types of objects, a constant number of membranes, and evolution rules of size $O(m^2)$.* □

4 Multiplication

4.1 Input and Output

Input of the multiplication is a pair of two m-bit binary numbers, V_x and V_y, and the two numbers are stored O_x and O_y, which are the same set of memory objects as input for the addition.

We also assume that a result of the multiplication is a $2m$-bit binary number V_p, i.e., $V_p = V_x \times V_y$, and V_p is stored in the following set of memory objects such that $V_p = \sum_{j=0}^{2m-1} V_{p,j} \times 2^j$.

$$\{\langle A_p, B_j, V_{p,j}\rangle \mid 0 \le j \le 2m-1\}$$

4.2 An asynchronous P system for multiplication

We describe an overview of the asynchronous P system Π_{MUL} that computes the multiplication. We first explain m binary numbers of $2m$ bits, $V_z, V_{z+1}, \cdots V_{z+(m-1)}$, which are used in the computation. In the first step of the computation, memory objects that denote each V_{z+k} are created so that $V_{z+k} = 0$. Then, each V_{z+k} is set to a value that denotes a product of V_x and k-th bit of V_y in the

Table 2: An example of V_{z+k}.

V_x	1011
V_y	1101
V_z	00001011
V_{z+1}	00000000
V_{z+2}	00101100
V_{z+3}	01011000
V_p	10001111

next step. More precisely, V_{z+k} is set as follows.

$$V_{z+k} = \begin{cases} 0 & (V_{y,k} = 0) \\ V_x \times 2^k & (V_{y,k} = 1) \end{cases}$$

Table 2 shows an example of V_{z+k} in case of $V_x = (1011)_2$ and $V_y = (1101)_2$. As shown in Table 2, the product V_p is given as the sum of $V_z, V_{z+1}, \cdots V_{z+(m-1)}$, i.e., $V_p = \sum_{k=0}^{m-1} V_{z+k}$.

Using the above idea, the computation of the asynchronous P system is executed in 4 steps given below.

Step 1 : Create memory objects that denote $V_z, V_{z+1}, \cdots, V_{z+(m-1)}$. (All values of the memory objects are set to 0.)

Step 2 : Compute V_{z+k} for each k $(0 \leq k \leq m-1)$.

Step 3 : Compute the sum of $V_z, V_{z+1}, \cdots, V_{z+(m-1)}$, and store the sum in V_p.

Step 4 : Output memory objects that denote V_p to the environment.

We now explain each set of evolution rules that realize the above 4 steps. The membrane structure used in the computation is skin membrane only.

In Step 1, the following sets of objects are asynchronously created from input memory objects. O_z is a set of memory objects that denote the binary numbers, $V_z, V_{z+1}, \cdots, V_{z+(m-1)}$, and O_H is a set of objects such that each $\langle H_{k,0} \rangle$ starts computation of V_{z+k}.

$$\begin{aligned} O_z &= \{\langle A_{z+k}, B_j, 0 \rangle \mid 0 \leq k \leq m-1, 0 \leq j \leq 2m-1\} \\ O_H &= \{\langle H_{k,0} \rangle \mid 0 \leq k \leq m-1\} \end{aligned}$$

The following set of evolution rules is applied to create the above two sets of objects for input memory objects.

$$\begin{aligned} R_1 &= R_{1,1} \cup R_{1,2} \\ R_{1,1} &= \{\langle A_x, B_0, V_{x,0} \rangle \rightarrow \langle A_{x'}, B_0, V_{x,0} \rangle \langle Z_{0,0} \rangle \mid V_{x,0} \in \{0,1\}\} \\ R_{1,2} &= \{\langle Z_{k,l} \rangle \rightarrow \langle A_{z+k}, B_l, 0 \rangle \langle Z_{k,l+1} \rangle \mid 0 \leq k \leq m-1, 0 \leq l \leq 2m-1\} \\ &\cup \{\langle Z_{k,2m} \rangle \rightarrow \langle Z_{k+1,0} \rangle \langle H_{k,0} \rangle \mid 0 \leq k \leq m-2\} \\ &\cup \{\langle Z_{m-1,2m} \rangle \rightarrow \langle H_{m-1,0} \rangle\} \end{aligned}$$

(Since object $\langle A_x, B_0, V_{x,0} \rangle$ triggers the computation, the object is transformed into the other object, $\langle A_{x'}, B_0, V_{x,0} \rangle$, after start of the computation.)

In Step 2, each V_{z+k} is computed applying the following set of evolution rules. In each evolution rule, an object $\langle H_{k,j} \rangle$ triggers the computation of an 1-bit product of $V_{x,j}$ and $V_{y,k}$, and the product is stored in $V_{z+k,j+k}$.

$$\begin{array}{rcl}
R_2 & = & R_{2,1} \cup R_{2,2} \cup R_{2,3} \\
R_{2,1} & = & \{\langle A_x, B_j, V_{x,j}\rangle\langle A_y, B_k, V_{y,k}\rangle\langle A_{z+k}, B_{j+k}, 0\rangle\langle H_{k,j}\rangle \\
 & & \quad \to \langle A_x, B_j, V_{x,j}\rangle\langle A_y, B_k, V_{y,k}\rangle\langle A_{z+k}, B_{j+k}, V_{x,j} \wedge V_{y,k}\rangle\langle H_{k,j+1}\rangle \\
 & & \quad \mid\ 0 \le k \le m-1, 1 \le j \le m-1, V_{x,j} \in \{0,1\}, V_{y,k} \in \{0,1\}\} \\
 & & \cup\{\langle A_{x'}, B_0, V_{x,0}\rangle\langle A_y, B_k, V_{y,k}\rangle\langle A_{z+k}, B_k, 0\rangle\langle H_{k,0}\rangle \\
 & & \quad \to \langle A_{x'}, B_0, V_{x,0}\rangle\langle A_y, B_k, V_{y,k}\rangle\langle A_{z+k}, B_k, V_{x,0} \wedge V_{y,k}\rangle\langle H_{k,1}\rangle \\
 & & \quad \mid\ 0 \le k \le m-1,\ V_{x,0} \in \{0,1\}, V_{y,k} \in \{0,1\}\} \\
R_{2,2} & = & \{\langle H_{0,m}\rangle \to \langle F_0\rangle\langle C_0, B_0, 0\rangle\} \cup \{\langle F_{k-1}\rangle\langle H_{k,m}\rangle \to \langle F_k\rangle\langle C_k, B_0, 0\rangle \mid 1 \le k \le m-1\} \\
R_{2,3} & = & \{[\langle F_{m-1}\rangle] \to [\ ]^+\}
\end{array}$$

Since our P system is asynchronous, the above computation may be executed in parallel for k, or may be executed sequentially. We detect the end of the computation using objects $\langle H_{k,m}\rangle$, which is created at the end of the computation for k. The detection is executed applying a set of evolution rules, $R_{2,2}$. Two objects, $\langle F_k\rangle$ and $\langle C_k, B_0, 0\rangle$, are created when the computation for V_{z+k} is finished. (The object $\langle C_k, B_0, 0\rangle$ is used in the next step as a carry bit for k-th addition.) Then, object $\langle F_{m-1}\rangle$, which is created at the end of Step 2, triggers Step 3 applying a set of evolution rules, $R_{2,3}$. The membrane is polarized $+$, and the polarization enables application of evolution rules for Step 3.

In Step 3, the sum of $V_z, V_{z+1}, \cdots, V_{z+(m-1)}$ is computed as follows. We first compute addition of each consecutive pair of the values, V_{z+2k} and V_{z+2k+1} ($0 \le k \le \frac{m}{2}$), and store the result of the addition in V_{z+m+k}. The addition is repeated using a traditional binary tree method. Then, the sum of $V_z, V_{z+1}, \cdots, V_{z+(m-1)}$ is obtained after the repetition, and the sum is stored in memory objects that denote V_{z+2m-2}.

The addition is executed applying the following set of evolution rules[1]. An object $\langle C_{2m-1}, B_0, 0\rangle$ is created at the end of Step 3, and triggers Step 4.

$$\begin{array}{rcl}
R_3 & = & R_{3,1} \cup R_{3,2} \\
R_{3,1} & = & \{\ [\langle C_{2i+1}, B_0, 0\rangle]^+ \to [\ ]^+ \mid\ 0 \le i \le m-2\} \\
 & & \cup\ \{\ [\langle C_{2i}, B_{2m}, 0\rangle]^+ \to \langle C_{i+m}, B_0, 0\rangle]^+ \mid 0 \le i \le m-1\} \\
R_{3,2} & = & \{[\langle A_{z+2i}, B_j, 0\rangle\langle A_{z+2i+1}, B_j, 0\rangle\langle C_{2i}, B_j, 0\rangle]^+ \to [\langle A_{z+i+m}, B_j, 0\rangle\langle C_{2i}, B_{j+1}, 0\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 0\rangle\langle A_{z+2i+1}, B_j, 1\rangle\langle C_{2i}, B_j, 0\rangle]^+ \to [\langle A_{z+i+m}, B_j, 1\rangle\langle C_{2i}, B_{j+1}, 0\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 1\rangle\langle A_{z+2i+1}, B_j, 0\rangle\langle C_{2i}, B_j, 0\rangle]^+ \to [\langle A_{z+i+m}, B_j, 1\rangle\langle C_{2i}, B_{j+1}, 0\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 1\rangle\langle A_{z+2i+1}, B_j, 1\rangle\langle C_{2i}, B_j, 0\rangle]^+ \to [\langle A_{z+i+m}, B_j, 0\rangle\langle C_{2i}, B_{j+1}, 1\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 0\rangle\langle A_{z+2i+1}, B_j, 0\rangle\langle C_{2i}, B_j, 1\rangle]^+ \to [\langle A_{z+i+m}, B_j, 1\rangle\langle C_{2i}, B_{j+1}, 0\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 0\rangle\langle A_{z+2i+1}, B_j, 1\rangle\langle C_{2i}, B_j, 1\rangle]^+ \to [\langle A_{z+i+m}, B_j, 0\rangle\langle C_{2i}, B_{j+1}, 1\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 1\rangle\langle A_{z+2i+1}, B_j, 0\rangle\langle C_{2i}, B_j, 1\rangle]^+ \to [\langle A_{z+i+m}, B_j, 0\rangle\langle C_{2i}, B_{j+1}, 1\rangle]^+, \\
 & & [\langle A_{z+2i}, B_j, 1\rangle\langle A_{z+2i+1}, B_j, 1\rangle\langle C_{2i}, B_j, 1\rangle]^+ \to [\langle A_{z+i+m}, B_j, 1\rangle\langle C_{2i}, B_{j+1}, 1\rangle]^+ \\
 & & \mid\ 0 \le i, j \le m-1\}
\end{array}$$

Since the addition can be executed in parallel for all pairs, we can execute the above addition for all pairs in $O(m)$ parallel steps, and can also execute the addition in $O(m^2)$ sequential steps.

Finally, the memory objects, which denote the sum, are sent out from the skin membrane applying the following set of rules in Step 4.

$$\begin{array}{rcl}
R_4 & = & R_{4,1} \cup R_{4,2} \\
R_{4,1} & = & \{[\langle A_{z+2m-2}, B_0, V_{z+2m-2,0}\rangle\langle C_{2m-1}, B_0, 0\rangle]^+ \to [\langle A_p, B_0, V_{z+2m-2,0}\rangle\langle G_1\rangle]\}
\end{array}$$

[1]Although the evolution rule is a little complicated, a basic idea of the rule is the same as the rule for addition in Section 3.

$$\begin{aligned} & \cup\{\langle A_{z+2m-2}, B_j, V_{z+2m-2,j}\rangle\langle G_j\rangle \rightarrow \langle A_p, B_j, V_{z+2m-2,j}\rangle\langle G_{j+1}\rangle \mid 1 \leq j \leq 2m-1\} \\ R_{4,2} \quad = \quad & \{[\langle A_p, B_j, V_{p,j}\rangle] \rightarrow [\ ]\langle A_p, B_j, V_{p,j}\rangle \mid 0 \leq j \leq m-1, V_{p,j} \in \{0,1\}\} \end{aligned}$$

We now summarize P system Π_{MUL} that asynchronously computes multiplication of two binary numbers of m bits.

$$\Pi_{MUL} = (O, \mu, \omega, R),$$

- O is a set of objects used in the evolution rule R.
- $\mu = [\ \ ]_1$
- $\omega = O_x \cup O_y$
- $R = R_1 \cup R_2 \cup R_3 \cup R_4$

Figure 5 illustrates an execution of the P system Π_{MUL}. In this example, two input numbers are 1011 and 1101. Two set of objects that denote the input numbers are given from the outside region into the skin membrane. Then, a set of evolution rule, R_1 is applied, and memory objects are created for the next step.

In the next step, products of 1011 and each bit of 1101 are computed using a set of evolution rule R_2, and the results are stored in memory objects created in the first step.

In the third step, sum of four results, which are obtained in the second step, is computed using a set of evolution rule R_3. The third step is executed in two sub-steps. In the first sub-step, two sums of two pairs of values stored in (A_z, A_{z+1}) and (A_{z+2}, A_{z+3}) are computed in parallel, and the sums are stored in A_4 and A_5. Then, the sum of sums obtained in the first sub-step is computed in the second sub-step, and the result is stored in A_6 at the end of the step.

Finally the result is sent out from the membrane using a set of evolution rule R_4.

4.3 Complexity

We now consider complexity of the P system Π_{MUL}. Step 1 and Step 2 can be executed in $O(m)$ parallel steps or $O(m^2)$ sequential steps. Step 3 can be executed in $O(m \log m)$ parallel steps or $O(m^3)$ steps because the addition for all pairs in Step 3 is executed in $O(m)$ parallel steps or $O(m^2)$ sequential steps, and the addition is repeated $O(\log m)$ times in parallel or $O(m)$ times sequentially. Therefore, we obtain the following theorem for Π_{MUL}.

Theorem 2 *The P system Π_{MUL}, which computes multiplication of two binary numbers of m bits, works in $O(m \log m)$ parallel steps or $O(m^3)$ sequential steps using $O(m^2)$ types of objects, a constant number of membranes, and evolution rules of size $O(m^2)$.* □

5 Factorization

We finally propose a P system that computes a factorization of a positive integer of m bits. Although two standard P systems [10, 12] have been proposed for computing the same factorization in a polynomial number of steps, their P system cannot work in asynchronous manner. Our P system works in both of maximally parallel and asynchronous manners, and computes the factorization in $O(m \log m)$ steps in maximally parallel manner.

We assume that input of the P system is a positive integer V_n, which is a product of two prime factors. We also assume that output of the P system is a two primes V_p and V_q such that $V_n = V_p \times V_q$. The input and output are given in the same sets of memory objects as input and output for the addition.

Since a basic idea of the P system is the same as [10, 12], we describe an outline of P system Π_{FACT}, which computes factorization of a positive integer. The P system mainly consists of three nested membranes. The outermost membrane is the skin membrane, and the second membrane,

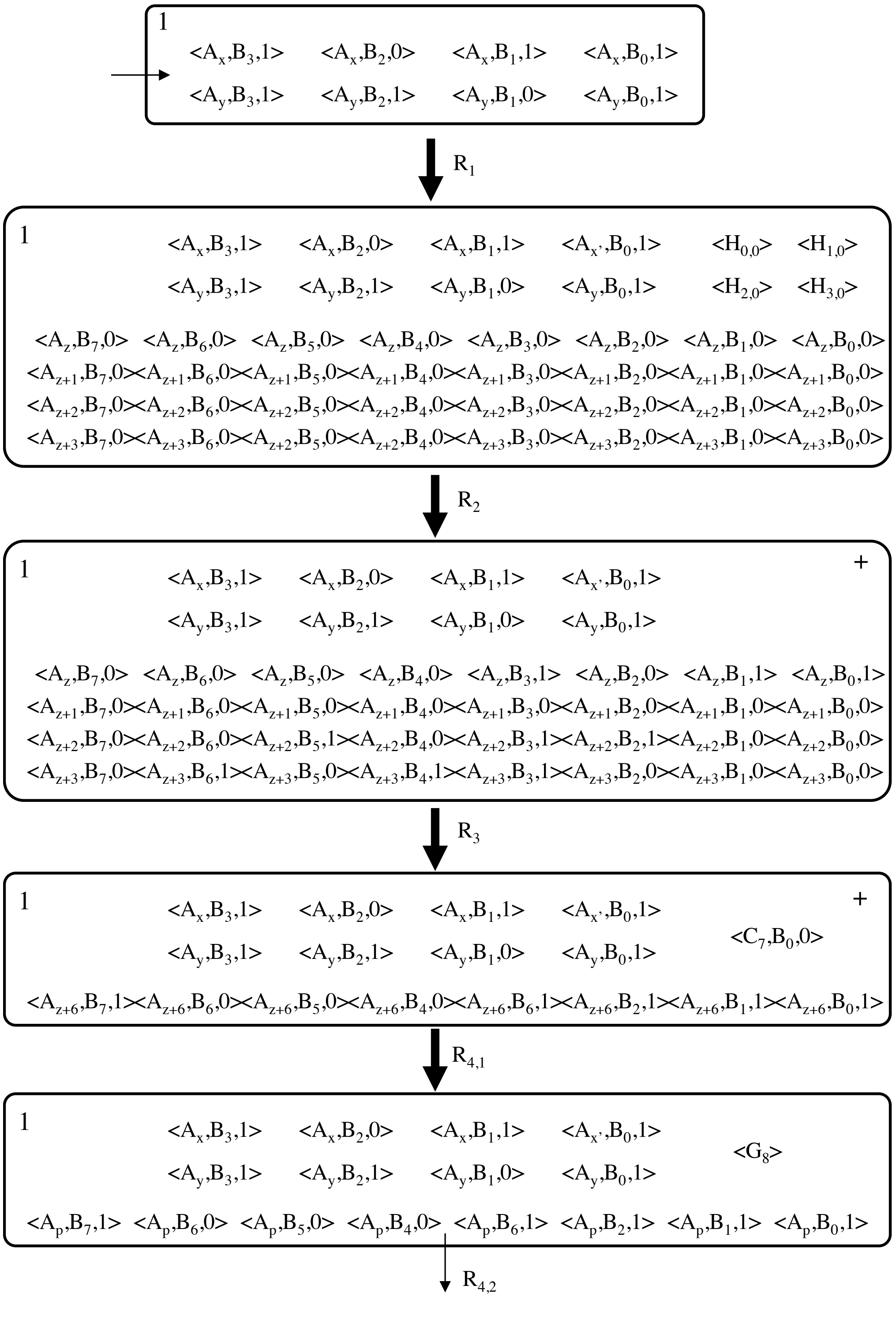
1
<Ax,B3,1> <Ax,B2,0> <Ax,B1,1> <Ax,B0,1>
<Ay,B3,1> <Ay,B2,1> <Ay,B1,0> <Ay,B0,1>
R1
R2
R3
R4,1
R4,2
<H0,0> <H1,0> <H2,0> <H3,0>
<C7,B0,0>
<G8>
+

Figure 5: An example of execution of Π_{MUL}.

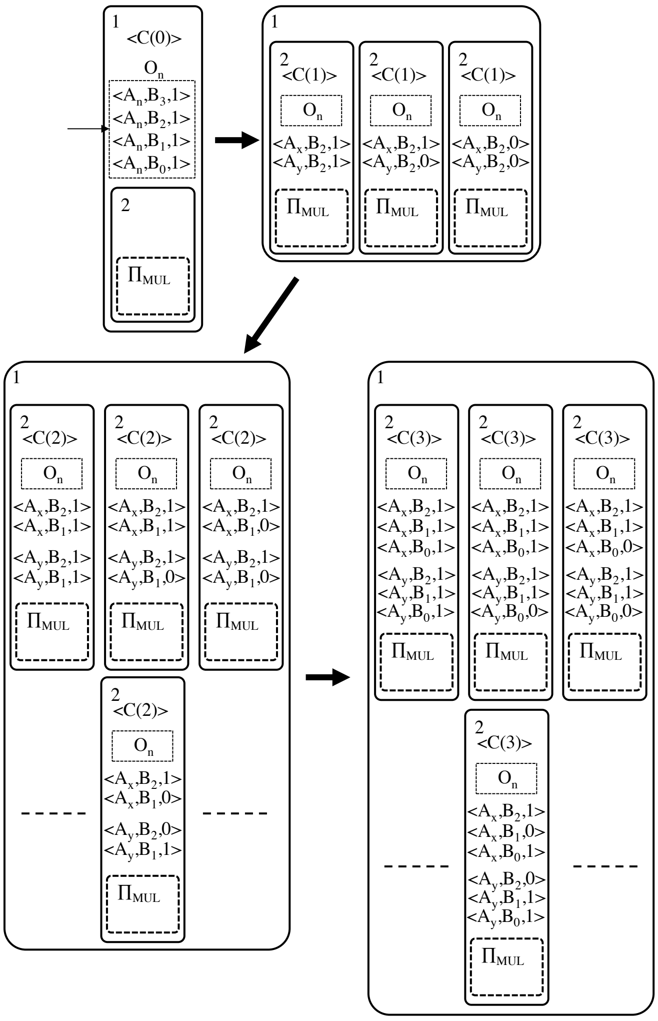

Figure 6: An outline of a generation phase of Π_{FACT}.

which is called a *candidate* membrane, is contained in the skin membrane. The candidate membrane also contains a *multiplication* membrane, which is P system Π_{MUL} described in Section 4.

The computation is executed in the following two phases.

(i) Generation phase: Using division rules, copies of *candidate* membrane are generated in the skin membrane. Each *candidate* membrane contains memory objects that denote a pair of integers, which are candidates for two factors. The division is executed so that all membranes contains distinct pairs of integers.

(ii) Verification and output phase: In each *candidate* membrane, a product of the pairs of integers is computed in *multiplication* membrane. Then, if the product is the same as the input integer, the *candidate* membrane is dissolved, and the memory objects are sent out from the skin membrane as output of the P system.

Figures 6 and 7 illustrate outlines of a generation phase and a verification and output phase, respectively. In Figure 6, a set of objects O_n, which denotes input binary number 1111, is given from the outside of a skin membrane. Then, the objects are sent in the inner *candidate* membrane, and the inner membrane is repeatedly divided so that each membrane contains memory objects that denote a pair of integers x, y such that $x > y$.

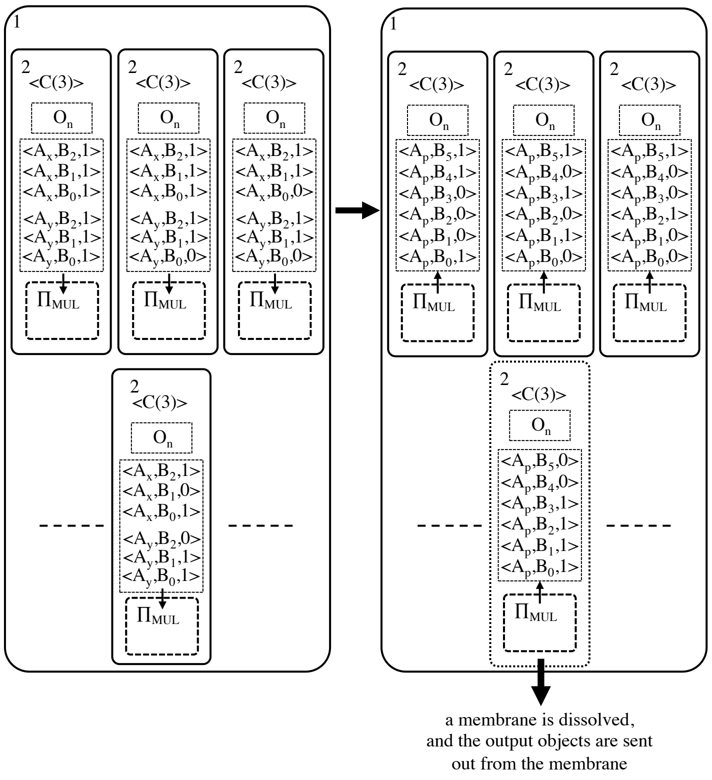

Figure 7: An outline of a verification phase of Π_{FACT}.

In Figure 7, each pair of the integers is sent into the inner membrane, which is a P system Π_{MUL}, and then, a product of the two integers is sent out from the inner membrane. The *candidate* membrane is dissolved if the product is equal to the input integer, and output objects are sent out from the skin membrane.

In the computation based on the above idea, $O(4^m)$ *candidate* membranes are generated in $O(m)$ parallel steps or $O(m^2)$ sequential steps in the first phase. Since a product of the two candidates is computed in each *candidate* membrane in $O(m \log m)$ parallel steps or $O(m^2 \log m)$ sequential steps, the total number of steps executed in Π_{FACT} is $O(m \log m)$ parallel steps or $O(4^m \times m^2 \log m)$ sequential steps. Since the number of objects and evolution rules used in the system are $O(m^2)$, we obtain the following theorem for Π_{FACT}.

Theorem 3 *The P system Π_{FACT}, which computes factorization of a positive number of m bits, works in $O(m \log m)$ parallel steps or $O(4^m \times m^2 \log m)$ sequential steps using $O(m^2)$ types of objects, $O(4^m)$ membranes, and evolution rules of size $O(m^2)$.* □

6 Conclusions

In the present paper, we have proposed asynchronous P systems that compute two basic arithmetic operations and factorization. The first and second asynchronous P systems compute addition and multiplication in a polynomial number of steps. We have also proposed the P system for factorization, and showed the P system works in a polynomial number of steps in maximally parallel manner, and also works in asynchronous manner.

In the future research, we will examine the application of the proposed asynchronous P systems for another numerical problems.

Acknowledgements

This work was supported by Grant-in-Aid for Scientific Research (C) (24500019).

References

[1] M. Cavaliere and V. Deufemia. Further results on time-free P systems. *International Journal of Foundations of Computer Science*, 17(1):69–90, 2006.

[2] M. Cavaliere and D. Sburlan. Time-independent p systems. In *International workshop on Membrane Computing*, pages 239–258, 2005.

[3] G. Ciobanu, L. Q. Pan, G. Păun, and M. J. Pérez-Jiménez. P systems with minimal parallelism. *Theoretical Computer Science*, 378(1):117–130, 2007.

[4] G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez. *Applications of Membrane Computing.* Springer, 2006.

[5] Z. Dang and O. H. Ibarra. On one-membrane P systems operating in sequential mode. *International Journal of Foundations of Computer Science*, 16(5):867–881, 2005.

[6] R. Freund. Asynchronous P systems and P systems working in the sequential mode. In *International workshop on Membrane Computing*, pages 36–62, 2005.

[7] R. Freund, A. Leporati, M. Oswald, and C. Zandron. Sequential P systems with unit rules and energy assigned to membranes. *Machines, Computations, and Universality*, 3354:200–210, 2005.

[8] 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*, 22(3):547–564, 2011.

[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*, 9(9):673–678, 2005.

[10] A. Kambe and A. Fujiwara. Arithmetic operations and factorization in membrane computing. In *Proceedings of Workshop on Algorithms and Computation*, pages 124–131, 2010.

[11] A. Leporati, C. Zandron, and M. A. Gutierrez-Naranjo. P systems with input in binary form. *International Journal of Foundations of Computer Science*, 17(1):127–146, 2006.

[12] A. Leporati, C. Zandron, and G. Mauri. Solving the factorization problem with P systems. *Progress in Natural Science*, 17(4):471–478, 2007.

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

[14] M. J. Pérez-Jiménez and A. Riscos-Núñez. A linear-time solution to the knapsack problem using P systems with active membranes. *Membrane Computing*, 2933:250–268, 2004.

[15] M. J. Pérez-Jiménez and A. Riscos-Núñez. Solving the subset-sum problem by P systems with active membranes. *New Generation Computing*, 23(4):339–356, 2005.

[16] M. J. Pérez-Jiménez and F.J. Romero-Campero. Solving the BIN PACKING problem by recognizer P systems with active membranes. In *The Second Brainstorming Week on Membrane Computing*, pages 414–430, 2004.

[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)*, E95-D(3), 2012.

[18] C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In *Unconventional Models of Computation*, pages 289–301, 2000.