PLOS ONE

RESEARCH ARTICLE

An arithmetic operation P system based on symmetric ternary system

Hai Nan[1], **Jie Zhang**[1], **Ping Guo**[2]*, **Jiqiao Jiang**[1], **Xu Zhang**[3]

1 School of Computer Science and Engineering, Chongqing University of Technology, Chongqing, Chongqing, China, **2** School of Artificial Intelligence and Big Data, Chongqing Metropolitan College of Science and Technology, Chongqing, Chongqing, China, **3** School of Big Data and Computer Science Engineering, Chongqing College of Mobile Communication, Chongqing, Chongqing, China

* guoping_cqck@cqcst.edu.cn

Citation: Nan H, Zhang J, Guo P, Jiang J, Zhang X (2024) An arithmetic operation P system based on symmetric ternary system. PLoS ONE 19(11): e0312778. https://doi.org/10.1371/journal.pone.0312778

Editor: Sameer Sheshrao Gajghate, GH Raisoni College of Engineering and Management Pune, INDIA

Received: July 25, 2024

Accepted: October 13, 2024

Published: November 1, 2024

Data Availability Statement: All data has been included in the manuscript.

Funding: This research was funded by the Science and Technology Research Program of Chongqing Municipal Education Commission (Grant No. KJQN201901133), the Chongqing Basic Research and Frontier Exploration Project (Chongqing Natural Science Foundation) under Grant (CSTB2022NSCQ-MSX0493), and National Natural Science Foundation of China (Grant No. 62141201).

Abstract

Nowadays, electronic computers use a "*binary*" numbering system, as opposed to "*ternary*" logic, which is closer to the way the human brain thinks. In this paper, the symmetric ternary system is applied to membrane computing for the first time. By combining the symmetric ternary system with membrane computing, this paper provides a more suitable arithmetic operation method for bio-computers, which breaks through the limitations of the traditional binary system in complex operations, and has a great potential for application in artificial intelligence and automatic learning in particular. The P System we designed include: Π^{+} for symmetric ternary addition, Π* for symmetric ternary multiplication, and $\Pi^{/}$ for symmetric ternary division. The operation process of each P System was explained through examples, and their feasibility and effectiveness were verified through simulation software, UPSimulator. The system we designed can be further applied to symmetric ternary applications.

1 Introduction

Existing computers use a "binary" numbering system, which, despite the simplicity of its computational rules, is not a perfect representation of what humans really think. In contrast, "ternary" logic is much closer to the way the human brain thinks.

Ternary is the base 3 for the system, generally has two forms of expression: one is to "0", "1", "2" as the basic character form of expression. One is a representation with "-1", "0", "1" as the base character, and this representation is also known as symmetric or balanced ternary. In general, we do not have only "true" and "false" answers to questions, but also "I don't know". In symmetric triadic logic, the symbol "1" represents "true"; the symbol "-1" represents "false"; the symbol "0" represents "I don't know". Obviously, this logical expression is more in line with the development trend of computers in artificial intelligence, which provides the possibility of fuzzy arithmetic and autonomous learning for computers. The logic of symmetric ternary is usually applied to decision-making [1], such as voting with "yes", "no", or "abstain"; trading with "buy", "sell", or "wait-and-see"; double-entry bookkeeping reflects the thinking of symmetric ternary; SQL database system adopts three-valued logic, which is the application of symmetric ternary.

Competing interests: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

However, ternary logic is not a new emphasis. Ternary computers have long had a precedent in the history of computer development. As early as the 1950s and 1960s, a group of researchers at Moscow State University designed the first ternary computers in the history of mankind, "Сетунь" and "Сетунь 70". The "Сетунь" computers used symmetric trigonometry instead of normal trigonometry [2]. Symmetric ternary logic circuits are not only faster and more reliable than binary logic circuits, but also require less equipment and power. One of the characteristics of symmetric ternary code is symmetry, i.e., the consistency of the opposite numbers, so that, unlike binary code, there is no concept of an "unsigned number". As a result, the architecture of a ternary computer is much simpler, more stable, and more economical. The instruction system is also easier to read and very efficient. At the same time, symmetric ternary can represent integers more naturally than binary, with fewer integer digits of smaller absolute value (omitting the zero before the first non-zero digit). The numbers it records can express the full range of integers, and the introduction of "-1" eliminates the need for an extra minus sign for negative numbers. Its corresponding logic circuits are "negative voltage", "zero voltage" and "positive voltage".

As computer technology continues to advance, symmetric ternary logic has once again attracted the attention of the scientific community. As chips are made smaller and smaller, semiconductors are gradually moving closer to the realm of quantum. Difficult problems like quantum tunneling, where we might have to put in a very large amount of effort to possibly improve efficiency a little bit, might have to start opening up other paths. And ternary, right now, is being resurrected in forms other than electronic computers. The electronic computer has only two base states, on and off. But photonic computers, there are light intensity, wavelength, phase, propagation direction and polarization of five states. Professor Yi Jin of Shanghai University, starting from the basic principles of constructing computers and the basic characteristics of light, for the first time combined light intensity and polarization direction to represent the three-valued information, and utilized the spinning effect of liquid crystals and polarizers to realize the interconversion and migration of the three optical states, which put forward a brand-new theory of optical computers—Ternary Optical Computer (referred to as the TOC) [3]. In 2019, Chinese physicist Guangchan Guo and his team successfully completed the transmission of a ternary quantum signal called "qutrit", which is the first successful ternary study by scientists in the quantum field [4].

Meanwhile, membrane computing has gradually become a popular research area in bio-computing. Membrane computing (also known as P System) is a new branch of natural computing, which is a new model of computation based on the abstraction of the structure and function of living cells and the collaboration of cell population such as tissues and organs. It is a computational model proposed by the Professor Gh.Păun in 1998 [5]. After Gh.Păun published his paper "Computing with membranes" in 2000 [6], it marked the birth of membrane computing as a research field. Since its introduction, membrane computing has attracted extensive attention from the scientific community, covering a wide range of disciplines or fields such as computer graphics and linguistics [7], biology [8], automation [9], and economics [10], and has rapidly evolved into a field of scientific research with great potential, and its development provides a rich computational framework for bio-computing.

With the intensive research on membrane computing, several studies have been devoted to the development and optimization of P System for arithmetic operations. Adrian Atanasiu designed arithmetic cell-like P System [11]. G.Ciobanu [12] designed arithmetic P System based on natural coding to realize arithmetic operations, which greatly simplified the membrane system structure. Ping Guo et al. [13] designed multi-layer membrane P System to realize arithmetic operations and reduce the computational complexity. Haiyan Zhang et al. [14] designed a single-layer membrane P System to realize arithmetic operations, which further

simplifies the membrane structure and improves the computational efficiency. Minghong Luo et al. [15] designed a multi-layer membrane P System to realize arithmetic operations with signed numbers. Ping Guo et al. [16–18] designed single-layer membrane P System to realize expression evaluation in the integer domain. Hong Zhang et al. [19] implemented basic arithmetic operations in the domain of rational numbers using P System. Kong, Y. et al. [20] investigated fundamental problems in fraction representation and arithmetic-fraction simplification. However all the above studies are based on binary or decimal.

While most research is still focused on binary and decimal systems, the potential of ternary is gradually emerging.Symmetric ternary is used in a number of applications due to its unique properties. Inspired by the balanced-ternary concept, Ji L et al. [21] demonstrates the reconfigurable generation of order-controllable vortices via cascaded N-layer meta surfaces. Faghih E et al. [22], for the first time, considers balanced ternary advantages to achieve a more efficient design for quantum multipliers as the main component in arithmetic blocks.

There are also many scientists who have devised arithmetic operations related to symmetric ternary. Ratan Kumar et al. [23] designs ternary logic circuits for nanoelectronics applications, the digital multiplier circuit is developed using Pseudo n-type carbon nanotube field effect transistors (CNTFETs). Based on the parallel carry-free TW-MSD adder, Yunfu S et al. [24] proposed a parallel R4-MSD square root algorithm, which is designed and implemented on the protype SD16 of ternary optical computer. Malik A et al. [25] proposes carbon nanotube field effect transistor (CNTFET)-based 'exact' and 'approximate' ternary full adders (TFA). Vudadha C [26] presents a new methodology to implement ternary Conditional Sum Adders (CSA) using CNFETs.

Although ternary has shown its potential for applications in several fields, its combination with membrane computing is still under-explored. And balanced ternary may become the most suitable number system for bio-computers. The study of arithmetic P System based on symmetric triples for membrane computing is of great academic and practical importance for the realization of a general-purpose bio-computer. The innovations of this paper mainly include:

1) Applying the symmetric ternary number system to membrane computing and designing a symmetric ternary arithmetic operation cell-like P System.

2) Dynamically creating cell membranes to realize arbitrary digit ternary arithmetic operations.

3) The symmetric ternary arithmetic operation system designed in this paper is simulated in UPSimulator (UPS), which is a simulator proposed in [27]. And the idea and feasibility of the algorithm are verified by examples.

The rest of the paper is organized as follows, Section 2 introduces the biological basis of membrane computing, describes the definition of cell-like P System. And then briefly introduces symmetric ternary and its arithmetic rules. Section 3 designs and implements an arithmetic P System based on symmetric ternary and detailing the rule execution process. Section 4 gives examples to elaborate the execution flow of the rule and verifies the correctness of the rule design through experimental simulation on computer. Section 5 summarizes the work accomplished in this paper and presents issues for future refinement.

2 Research foundation

2.1 Cell-like P system

The cell-like P System is one of the most basic and earliest proposed model of membrane computing [6], and an abstract schematic of the cell-like P System is shown in Fig 1. The

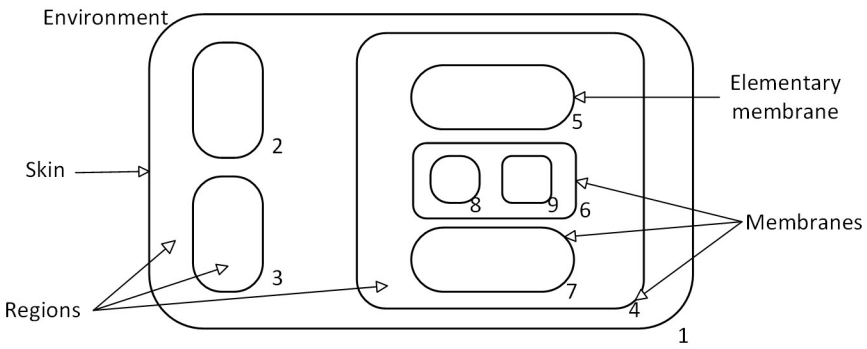

Fig 1. The structure of cell-like P system [6].

https://doi.org/10.1371/journal.pone.0312778.g001

membrane computing model divides a cell into multiple regions with a hierarchical structure, and the boundary of each region is the membrane. The outermost membrane, called the skin, separates the entire membrane system from its external environment, and the region outside the skin is the environment. If there are no other membranes within the membrane, it is called the basic membrane [28]. Each membrane represents a region; the region of a basic membrane is the space it contains; the region of a non-basic membrane refers to the space between the membrane itself and the membrane it directly contains. Regions contain objects represented by multi-sets, and objects evolve by executing reaction rules: objects are converted into other objects that can reach a certain membrane, which can also be dissolved or split. The execution of rules follows a nondeterministic and parallel character. The time when there are no rules to be executed in the region is called downtime, and the results of the computation are sent in the specified membrane or environment.

Membrane structures can be represented by generalized tables. A membrane is denoted by a pair of brackets '[]', with the subscripts of the brackets denoting the label of the membrane. The basic membrane i is denoted as $[_i]_i$; if membrane i contains membrane k inside, the membrane structure is denoted as $[_i[_k]_k]_i$. The membrane structure of Fig 1 can be represented by the generalized table $[1\ [2]_2\ [3]_3\ [4\ [5]_5\ [6\ [8]_8\ [9]_9]_6\ [7]_7]_4]_1$.

A cell-like P System of degree m ($m \geq 1$) is defined as formula (1) [6].

$$\Pi = (V, \mu, \omega_1, \ldots, \omega_m, R_1, \ldots, R_m, \rho_1, \rho_2, \ldots, \rho_m, i_0) \tag{1}$$

where:

1. V is a finite non-empty alphabet, whose elements are objects;
2. μ is a membrane structure containing m membranes, where m is called the degree of Π;
3. $\omega_i \in V^*$ $(1 \leq i \leq m)$, denotes the multiset of objects contained inside region i in the membrane structure μ. V^* is the set of arbitrary strings consisting of characters in V;
4. R_i $(1 \leq i \leq m)$ is a finite set of evolutionary rules inside region i in the membrane structure μ, the evolutionary rules are binary groups (u, v), usually written as $u \rightarrow v$, where, u is a string in V^+ and V^+ is a set of non-empty strings in V^*, $v = v'$ or $v = v'\delta$, here v' is a string on the set $\{a_{\text{here}}, a_{\text{out}}, a_{\text{in}_j} \mid a \in V, 1 \leq j \leq m\}$, δ is a special character that does not belong to V, when a rule contains δ, the membrane is dissolved after executing the rule, the length of u is known as the radius of the rule $u \rightarrow v$;

5. ρ_i $(1 \leq i \leq m)$, denotes a partial order relation in R_i;

6. i_o is a number between 1 and m where the output of results in Π.

In this paper, the initial grid refers to the P System that has not yet started the computation. When operands are sent into the P System, which triggers the rules to be executed, the computation starts. The P System at a certain time slice in the computation is called the configuration at that moment. As rules are executed, the configuration of the P System will change until there are no rules left to be executed.

In every membrane structure, the rules will be enforced according to the following two principles:

1) Uncertainty. The P System will follow the principle of uncertainty when executing rules, which means that when there are n evolutionary rules in the membrane that can be executed at the same time, the P System randomly selects some of the rules to be implemented and the objects in the system to be evolved and chooses the rule that governs this evolution in a non-deterministic way [28].

2) Maximum Parallelism. In the P System, each step of the computation follows the principle of maximum parallelism, which means that all the rules that can be executed must be executed at the same time.

2.2 Symmetric ternary

The symmetric ternary was inspired by Gauss's idea of the simplest set of weights. The simplest set of weights problem is as follows: How should the simplest set of weights be designed for weighing an object of any integer gram weight with weights on a balance. Usually when weighing an object with weights on a balance, the object to be weighed is placed on one side of the balance pan and the weights on the other side. Gauss proposed that the simplest set of weights is 1, 3, 9, 27, . . ., 3^n, . . . grams, and that the weights can be placed on either side of the balance pan when weighing an object. It can be shown that the formula for weighing an object of any integer gram weight with the simplest set of weights is expressed as follows [29]:

$$K = a_n 3^n + a_{n-1} 3^{n-1} + \cdots + a_1 3^1 + a_0 3^0. \tag{2}$$

Where K is any positive integer, 3^n, 3^{n-1}, . . ., 3^1, 3^0 is the weight of each weight, the coefficients a_n, a_{n-1}, . . ., a_1, a_0 is one of −1,0,1. "1" represents that the weight is placed on the other side of the balance pan of the object to be weighed, "−1" represents that the weight is placed on the same side of the object to be weighed. And "0" means that the weight does not participate in the weighing. The 3^n, 3^{n-1}, . . ., 3^1, 3^0 are used to represent the mass of the weights are viewed as bit-weights, and ignoring the bit-weights, any positive integer K can be expressed in the following form [29]:

$$K = a_n, a_{n-1}, \ldots, a_1, a_0. \tag{3}$$

Where a_n, a_{n-1}, . . ., a_1, a_0 is one of −1,0,1. To avoid confusion, −1 is generally denoted by T, and Z or z under special conditions. In this paper, we denote −1 by T. This representation of an arbitrary integer K by a string of numbers consisting of the various coefficients is called symmetric ternary.

Symmetric ternary system has many advantages, first of all, it has both positive and negative number elements, which can be expressed as positive or negative numbers by the same Eq (2). The sign of the first digit can be used to determine whether K is positive or negative; i.e., when the first digit is positive, K is positive, and when the first digit is negative, K is negative. The quadratic operations for symmetric ternary are also simple, and Tables 1–4 shows the quadratic rules for symmetric ternary.

Table 1. Addition in symmetric ternary.

+	TT	T0	T1	T	0	1	1T	10	11
11	0	1	1T	10	11	1TT	10T	1T1	10T
10	T	0	1	1T	10	11	1TT	1T0	-
1T	T1	T	0	1	1T	10	11	-	-
1	T0	T1	T	0	1	1T	-	-	-
0	TT	T0	T1	T	0	1	-	-	-
T	T11	TT	T0	T1	T	0	-	-	-
T1	T10	T11	TT	-	-	-	-	-	-
T0	T1T	T10	-	-	-	-	-	-	-
TT	T01	-	-	-	-	-	-	-	-

https://doi.org/10.1371/journal.pone.0312778.t001

Table 2. Subtraction in symmetric ternary.

-	TT	T0	T1	T	0	1	1T	10	11
TT	0	T	T1	T0	TT	T11	T10	T1T	T01
T0	1	0	T	T1	T0	TT	T11	T10	T1T
T1	1T	1	0	T	T1	T0	TT	T11	T10
1	10	1T	1	0	T	T1	T0	TT	T11
0	11	10	1T	1	0	T	T1	T0	TT
T	1TT	11	10	1T	1	0	T	T1	T0
T1	1T0	1TT	11	10	1T	1	0	T	T1
T0	1T1	1T0	1TT	11	10	1T	1	0	T
TT	I0T	1T1	1T0	1TT	11	10	1T	1	0

Note: Left column minus top row

https://doi.org/10.1371/journal.pone.0312778.t002

Table 3. Multiplication in symmetric ternary.

*	TT	T0	T1	T	0	1	1T	10	11
11	T11T	TT0	T10	TT	0	11	10T	110	1TT1
10	TT0	T00	T10	T0	0	10	1T0	100	-
1T	T01	T10	1T	1	0	1T	11	-	-
1	TT	T0	T1	T	0	1	-	-	-
0	0	0	0	0	0	0	-	-	-
T	11	10	1T	1	0	T	-	-	-
T1	10T	1T0	11	-	-	-	-	-	-
T0	110	100	-	-	-	-	-	-	-
TT	1TT1	-	-	-	-	-	-	-	-

https://doi.org/10.1371/journal.pone.0312778.t003

Table 4. Division in symmetric ternary.

/	TT	T0	T1	T	0	1	1T	10	11
TT	1	1.1	1T	11	-∞	TT	T1	T.T	T
T0	1.$\bar{T}$1	1	1.$\bar{1}$	10	-∞	T0	T.$\bar{T}$	T	T.1$\bar{T}$
T1	1.$\bar{T}$	1.T	1	1T	-∞	T1	T	T.1	T.$\bar{1}$
T	0.1$\bar{T}$	0.1	0.$\bar{1}$	1	-∞	T	0.$\bar{T}$	0.T	0.$\bar{T}$1
0	0	0	0	0	NaN	0	0	0	0
1	0.$\bar{T}$1	0.T	0.$\bar{T}$	T	+∞	1	0.$\bar{1}$	0.1	0.1$\bar{T}$
1T	T.$\bar{1}$	T.1	T	T1	+∞	1T	1	1.T	1.$\bar{T}$
10	T.1$\bar{T}$	T	T.$\bar{T}$	T0	+∞	10	1.$\bar{1}$	1	1.$\bar{T}$1
11	T	T.T	T1	TT	+∞	11	1T	1.1	

Note: Left column divided top row

https://doi.org/10.1371/journal.pone.0312778.t004

3 Arithmetic P system based on symmetric ternary system

3.1 Addition and subtraction

According to the definition of cell-like P System, an addition and subtraction arithmetic P System based on symmetrical ternary can be defined as:

$$\Pi^{+} = (V, \mu, \omega_1, \omega_{M_1}, R, \rho, i_0) \tag{4}$$

Where:

$V = \{a, b, c, T, 0, 1, s, f, E, Y\}$;

$\mu = [_1[_{M_1}]_{M_1}]_1$;

$\omega_1 = \{Y\}$;

$\omega_{M_1} = \{f\}$;

i_0 consists of M_1 and his submembrane to hold the output;

$\rho = 1, 2$;

$R = R_1 \cup R_{M_1}$;

$R_1 = \{r_1: (T \rightarrow (a, M_1), 1), r_2: (0 \rightarrow (b, M_1), 1), r_3: (1 \rightarrow (c, M_1), 1),$
$r_4: (sY \rightarrow E, 1), r_5: (E \rightarrow (E, M_1), 1)\}$;

$R_{M_1} = \{r_1 : (af \rightarrow T[f], 1), r_2 : (bf \rightarrow 0[f], 1), r_3 : (cf \rightarrow 1[f], 1),$
$r_4: (Ef \rightarrow E[f], 1), r_5: (a \rightarrow (a, \text{in}), 2), r_6: (b \rightarrow (b, \text{in}), 2), r_7: (c \rightarrow (c, \text{in}), 2),$
$r_8: (aE \rightarrow T(E, \text{in}), 1), r_9: (bE \rightarrow 0(E, \text{in}), 1), r_{10}: (cE \rightarrow 1(E, \text{in}), 1),$
$r_{11}: (0^2 \rightarrow 0, 1), r_{12}: (0T \rightarrow T, 1), r_{13}: (01 \rightarrow 1, 1), r_{14}: (T^2 \rightarrow 1(T, \text{in}), 1),$
$r_{15}: (T1 \rightarrow 0, 1), r_{16}: (1^2 \rightarrow T(1, \text{in}), 1)\}$;

In this case, the correspondence of objects is as follows. The augend numbers T, 0, 1 are represented as objects a, b, c when they enter membrane M_1 from membrane 1. Objects a, b, c reach to the membrane where object f is located, and are converted to objects T, 0 and 1 after reacting with f. In this way, the incoming objects a, b, and c will only react if they enter the membrane where f is located, thus enabling dynamic modeling and the storage of the augend numbers from low to high in membrane M_1 and its submembranes. Object s is used to indicate the end of the augend input and to generate E with Y in Membrane 1. The purpose of E is twofold: to signal that the system is ready to input the addend, and to convert the addend to T, 0, and 1 so that it can be added to the augend.

Let us assume that the augend is $a_n a_{n-1} \ldots a_1$ and the addend is $b_m b_{m-1} \ldots b_1$. Then we illustrate the use of the rules in Π^{+} by adding these two numbers.

(1) Input of the augend:

We input one bit of the augend every two time slices from low to high. First a_1 is sent into membrane 1.

- **Case 1**: $a_1 = T$. Executing rule r_1 in R_1, object T is converted to object a and sent into membrane M_1. Executing rule r_1 in R_M, object a and object f are consumed with T produced, a new membrane M_2 is created at the same time, and f is sent into membrane M_2.
- **Case 2**: $a_1 = 0$. Executing rule r_2 in R_1, object 0 is converted to object b and sent into membrane M_1. Executing rule r_2 in R_M, object b and object f are consumed with 0 produced, a new membrane M_2 is created, and f is sent into membrane M_2.
- **Case 3**: $a_1 = 1$. Executing rule r_3 in R_1, object 1 is converted to object c and is sent into membrane M_1. Executing rule r_3 in R_M, object c and object f are consumed with 1 produced, a new membrane M_2 is created, and f is sent into membrane M_2.

Object a_i $(1 < i < n)$ is sent into membrane 1, converted to object a, b, or c (R_1: $r_1 \sim r_3$), and enters membrane M_1, then continues into the inner membrane (R_M: $r_5 \sim r_7$) until it reaches membrane M_i where it is consumed with f and produces object T, 0, or 1 (R_M: $r_1 \sim r_3$).

The last object a_n is input to the system with s. Object s is used to indicate that the augend has been fully entered. Rule r_4 in R_1 is executed, object s is consumed with Y and E produced, indicating that the system is ready to input the addend, while E enters Membrane M_1.

(2) Input of the addend:

Object b_1 is sent into membrane 1.

- **Case 1**: $b_1 = T$, rule r_1 in R_1 is executed, object T is converted to a to enter membrane M_1. Object a is consumed with E (R_M: r_8), a is converted to T to be preserved in membrane M_1, and E enters the inner membrane M_2.
- **Case 2**: $b_1 = 0$, executing rule r_2 in R_1. Object 0 is converted to b and is sent into membrane M_1. Object b and E are consumed (R_M: r_9), generating 0 to remain in membrane M_1 and E is sent into membrane M_2.
- **Case 3**: $b_1 = 1$, executing rule r_3 in R_1. Object 1 is converted to c to enter M_1, then object c is consumed with E (R_M: r_{10}) and object 1 produced to stay in membrane M_1, and E enters membrane M_2.

The next object b_i $(1 < i \leq m)$ follows a similar pattern to b_1. It is sent into membrane 1 first, and then converted to object a or b or c (R_1: $r_1 \sim r_3$), entering membrane M_1. Object b_i continues into the inner membrane (R_M: $r_5 \sim r_7$) until it reaches membrane M_i. Then it is consumed with E to produce object T, 0 or 1 (R_M: $r_1 \sim r_3$).

(3) Addition:

The process of addition is simultaneous with the input of the addend. When b_1 reaches membrane M_1, the addition operation can be performed without having to wait for the addend to be fully input. Two numbers are added bit-wise from low to high and may produce carrying. The possibilities of $a_i + b_i$ $(1 \leq i \leq \min\{n, m\})$ in $a_n a_{n-1} \ldots a_1$ and $b_m b_{m-1} \ldots b_1$ are as follows:

- **Case 1**: $0 + 0 = 0$ (R_M: r_{11});
- **Case 2**: $0 + T = T$ (R_M: r_{12});
- **Case 3**: $0 + 1 = 1$ (R_M: r_{13});
- **Case 4**: $T + T = T1$ (R_M: r_{14});
- **Case 5**: $T + 1 = 0$ (R_M: r_{15});

- **Case 6**: $1 + 1 = 1T$ (R_M: r_{16}).

For Case 4 and Case 6 that have generated feeds, the high bit of the result is sent into the inner membrane and the low bit is left in the original membrane.

The result of the addition is saved in the membrane $M_1, \ldots, M_k$ ($k \geq 1$) from low to high. The example of addition in Section 4.1.1 provides a more concrete demonstration of the implementation of the rules. In the addition P System Π^+, symmetric ternary numbers "$n + m$" require at most $3n + 4m$ time slices for addition.

The subtraction P System simply changes the rule r_1 in R_1 to $(T \rightarrow (c, M_1))$, and r_3 to $(1 \rightarrow (a, M_1))$, the operations are all consistent with addition, so we won't go into too much detail here.

3.2 Multiplication

According to the definition of cell-like P System, a multiplication arithmetic P System based on symmetrical ternary can be defined as:

$$\Pi^* = (V, \mu, \omega_1, \omega_{M_1}, R, \rho, i_0) \tag{5}$$

Where:

$V = \{a, b, c, T, 0, 1, A, B, C, x, y, z, p, q, r, u, n, s, f, E, Y\}$;

$\mu = [_1[_{M_1}]_{M_1}]_1$;

$\omega_1 = \{Y\}$;

$\omega_{M_1} = \{f\}$;

i_0 consists of M_1 and his submembrane to hold the output;

$\rho = 0, 1, 2$;

$R = R_1 \cup R_{M_1}$;

$R_1 = \{r_1: (T \rightarrow (a, M_1), 2), r_2: (0 \rightarrow (b, M_1), 2), r_3: (1 \rightarrow (c, M_1), 2),$
$r_4: (Tn \rightarrow un(a, M_1), 1), r_5: (0n \rightarrow un(b, M_1), 1), r_6: (1n \rightarrow un(c, M_1), 1),$
$r_7: (sY \rightarrow En, 1), r_8: (E \rightarrow (E, M_1), 1), r_9: (u \rightarrow (u, M_1)|_T, 0),$
$r_{10}: (u \rightarrow (u, M_1)|_0, 0), r_{11}: (u \rightarrow (u, M_1)|_1, 0)\}$

$R_{M_1} = \{r_1 : (af \rightarrow A[f], 1), r_2 : (bf \rightarrow B[f], 1), r_3 : (cf \rightarrow C[f], 1),$
$r_4: (a \rightarrow (a, in), 2), r_5: (b \rightarrow (b, in), 2), r_6: (c \rightarrow (c, in), 2), r_7: (aE \rightarrow x(E, in), 1),$
$r_8: (bE \rightarrow y(E, in), 1), r_9: (cE \rightarrow z(E, in), 1), r_{10}: (Ax \rightarrow 1(px, in), 1),$
$r_{11}: (Bx \rightarrow 0(qx, in), 1), r_{12}: (Cx \rightarrow T(rx, in), 1), r_{13}: (Ay \rightarrow 0(py, in), 1),$
$r_{14}: (By \rightarrow 0(qy, in), 1), r_{15}: (Cy \rightarrow 0(ry, in), 1), r_{16}: (Az \rightarrow T(pz, in), 1),$
$r_{17}: (Bz \rightarrow 0(qz, in), 1), r_{18}: (Cz \rightarrow 1(rz, in), 1), r_{19}: (xf \rightarrow [f], 1),$
$r_{20}: (yf \rightarrow [f], 1), r_{26}: (ru \rightarrow C(u, in), 1), r_{27}: (0^2 \rightarrow 0, 1), r_{28}: (0T \rightarrow T, 1),$
$r_{29}: (01 \rightarrow 1, 1), r_{30}: (T^2 \rightarrow 1(T, in), 1), r_{31}: (T1 \rightarrow 0, 1), r_{32}: (1^2 \rightarrow T(1, in), 1)\}$

In this case, the correspondence of the objects is as follows.

The multiplicand numbers T, 0, and 1 are represented as objects a, b, and c when they enter membrane M_1 from membrane 1. They reach the membrane where object f is located, and are converted to objects A, B and C after reacting with f. They are converted to objects p, q, and r when they are multiplied by the multiplier and move toward the inner membrane.

The multiplier numbers enter the membrane M_1 also represented by the objects a, b, and c. When they arrive in the membrane where E is located, they are represented by the objects x, y, and z after reacting with E. Objects s, f, E, and Y act in the same way as addition. Object n is to control the generation of u, which converts the shifted multiplicands p, q, and r into the objects A, B, and C.

Let us assume that the multiplicand is $a_n a_{n-1}...a_1$ and the multiplier is $b_m b_{m-1}...b_1$. We illustrate the use of the rules in Π^* by multiplying two numbers.

(1) Input of the multiplicand:

We input one bit of the multiplicand from low to high every two time slices. First, object a_1 is sent into membrane 1.

- **Case 1**: $a_1 = T$, executing rule r_1 in R_1, object T is converted to a and is sent into membrane M_1. Object a is consumed with f (R_M: r_1), a is converted to A to stay in membrane M_1 and a new membrane M_2 is generated, and f enters into membrane M_2.
- **Case 2**: $a_1 = 0$, executing rule r_2 in R_1, object 0 is converted to b and is sent into membrane M_1. Object b is consumed with f (R_M: r_2), b is converted to B to stay in membrane M_1 and a new membrane M_2 is generated, and f enters into membrane M_2.
- **Case 3**: $a_1 = 1$, executing rule r_3 in R_1, object 1 is converted to c and is sent into membrane M_1. Object c is consumed with f (R_M: r_3), c is converted to C to stay in membrane M_1 and a new membrane M_2 is generated, and f enters into membrane M_2.

The rules for the execution of the multiplicand a_i ($1 < i < n$) are similar to the rules for a_1. Object a_i is sent into membrane 1 first, then it is converted to object a, b or c into membrane M_1. It continues into the inner membrane until it reaches membrane M_i where it is consumed with f to produce object A, B or C. The highest bit of multiplicand a_n is sent in membrane 1 with object s, which is used to indicate that the multiplicand has been completely imported. Rule r_7 in R_1 is applied, object s is consumed with Y to produce E and n, signaling that it is ready to send in the multiplier. At the same time, E is sent into membrane M_1, and n stays in membrane 1 for the production of u.

(2) Input of the multiplier:

Object b_1 is sent into membrane 1.

- **Case 1**: $b_1 = T$, rule r_4 in R_1 is applied, object T and n are consumed to produce u, n, and a, object a is sent into membrane M_1. Rule r_7 in R_M is applied, object a is converted to x to stay in membrane M_1, and E enters membrane M_2.
- **Case 2**: $b_1 = 0$, rule r_5 in R_1 is applied, object 0 and n are consumed to produce u, n, and b, object b is sent into membrane M_1. Rule r_8 in R_M is applied, object b is converted to y to stay in membrane M_1, and E enters membrane M_2.
- **Case 3**: $b_1 = 1$, rule r_6 in R_1 is applied, object 1 and n are consumed to produce u, n, and c, object c is sent into membrane M_1. Rule r_9 in R_M is applied, object c is converted to z to stay in membrane M_1, and E is sent into membrane M_2.

The multiplier b_i ($2 \leq i \leq m$) performs similar rules to b_1. Object b_i is sent into membrane 1, then it is converted to object a, b, or c into membrane M_1. It continues into the inner membrane until it reaches membrane M_i where it is consumed with E and produces object x, y, or z.

(3) Multiplication:

Multiplication of two numbers involves both multiplication and addition operations, and these rules are performed simultaneously. When object b_1 is sent into membrane M_1, it is prepared for multiplication operations with $a_n a_{n-1}...a_1$.

Multiplication possibilities of $b_1 \times a_1$ are as follows:

- **Case 1**: $T \times T = 1$ (R_M: r_{10});
- **Case 2**: $T \times 0 = 0$ (R_M: r_{13});

- **Case 3**: $T \times 1 = T$ (R_M: r_{16});
- **Case 4**: $0 \times T = 0$ (R_M: r_{11});
- **Case 5**: $0 \times 0 = 0$ (R_M: r_{14});
- **Case 6**: $0 \times 1 = 0$ (R_M: r_{17});
- **Case 7**: $1 \times T = T$ (R_M: r_{12});
- **Case 8**: $1 \times 0 = 0$ (R_M: r_{15});
- **Case 9**: $1 \times 1 = 1$ (R_M: r_{18}).

Taking the case of "$T \times T = 1$" as an example, Rule R_M: r_{10} signifies that object A is consumed with x to produce 1, which is retained in the original membrane. Object A is converted to p and sent into membrane M_2 along with object x. The conversion of object A to p and its movement into the inner membrane serves to: 1) Avoid confusion with the multiplicand object in the next layer of the membrane; 2) Allow the result of the multiplication to directly contribute to addition operations in the original membrane. After the input of b_2, object u will first be sent to membrane M_1 (R_M: $r_9 \sim r_{11}$), where u will convert p back to A (R_M: r_{24}) before b_2 is sent into membrane M_2, enabling b_2 to multiply with object A.

Object b_1 is sent to membrane M_2 to multiply by a_2. The result of the calculation is retained in the original membrane, a_2 is converted and sent into membrane M_3 along with the multiplicand b_1. This process continues until b_1 enters membrane M_n. After multiplication by a_n, the converted b_1 and a_n arrive at membrane M_{n+1}, where f is located. Object b_1 is consumed with f (R_M: $r_{19} \sim r_{21}$), producing a new membrane M_{n+2}, and f is sent into the new membrane to prevent spillage.

After the input of b_2, object u will be sent into membrane M_1 first (R_M: $r_9 \sim r_{11}$). Object u will convert the multiplicand from object p, q, or r to object A, B, or C (R_M: $r_{24} \sim r_{26}$) before b_2 is sent into membrane M_2. In membrane M_2, object b_2 multiplies with a_1, and the product is then added to the result of $b_1 \times a_1$ according to rules (R_M: $r_{27} \sim r_{32}$). b_2 then continues into the inner membrane for further reactions. The next objects b_i ($3 \leq i \leq m$) also react according to the earlier described rules.

The final result is stored sequentially in membranes $M_1, \ldots, M_k$ ($k \geq 1$). The multiplication example in Section 4.1.2 demonstrates the concrete implementation of the rules. In the multiplication P System Π^*, symmetric ternary numbers "$n \times m$" require at most $3n + 4m + 3$ time slices to complete the multiplication operation.

3.3 Division

According to the definition of cell-like P Systems, a division arithmetic P System based on symmetrical ternary can be defined as:

$$\Pi^{/} = (V, \mu, \omega_1, \omega_{M_1}, R, \rho, i_0) \tag{6}$$

Where:

$V = \{a, b, c, T, 0, 1, A, B, C, K, M, N, H, V, k, x, i, v, r, u, g, s, f, e, E, Y, X\}$;

$\mu = [_1[_{M_1}]_{M_1}]_1$;

$\omega_1 = \{Y\}$;

$\omega_{M_1} = \{f\}$;

i_0 consists of M_1 and his submembrane to hold the output;

$\rho = 0, 1, 2, 3, 4, 5$;

$R = R_1 \cup R_{M_1}$;
$R_1 = \{r_1: (T \to (a, M_1)|_Y, 1), r_2: (0 \to (b, M_1)|_Y, 1), r_3: (1 \to (c, M_1)|_Y, 1),$
$r_4: (T \to (c, M_1), 2), r_5: (0 \to (b, M_1), 2), r_6: (1 \to (a, M_1), 2),$
$r_7: (sY \to E(s, M_1), 1), r_8: (E \to (E, M_1), 1), r_9: (sK \to eK, 2),$
$r_{10}: (K \to Nk|_e, 1), r_{11}: (ke \to e(k, M_1), 1), r_{12}: (N \to M|_e, 1),$
$r_{13}: (M \to H|_e, 1), r_{14}: (H \to K|_e, 1), r_{15}: (xe \to X(V, in), 0),$
$r_{16}: (vX \to XV(k, in), 1), r_{17}: (V \to (V, in), 1), r_{18}: (kX \to X, 0)\}$
$R_{M_1} = \{r_1 : (af \to T[f], 1), r_2 : (bf \to 0[f], 1), r_3 : (cf \to 1[f], 1),$
$r_4: (a \to (a, in), 2), r_5: (b \to (b, in), 2), r_6: (c \to (c, in), 2),$
$r_7: (aE \to A(E, in), 1), r_8: (bE \to B(E, in), 1), r_9: (cE \to C(E, in), 1),$
$r_{10}: (0k \to T, (k, in)|_A, 3), r_{11}: (0k \to 0, (k, in)|_B, 3), r_{12}: (0k \to 1, (k, in)|_C, 3),$
$r_{13}: (Tk \to 1, (Tk, in)|_A, 3), r_{14}: (Tk \to T, (k, in)|_B, 3), r_{15}: (Tk \to 0, (k, in)|_C, 3),$
$r_{16}: (1k \to 0, (k, in)|_A, 3), r_{17}: (1k \to 1, (k, in)|_B, 3), r_{18}: (1k \to T, (1k, in)|_C, 3),$
$r_{19}: (0^2 \to 0, 2), r_{20}: (0T \to T, 2), r_{21}: (01 \to 1, 2), r_{22}: (T^2 \to 1(T, in), 2),$
$r_{23}: (T1 \to 0, 2), r_{24}: (1^2 \to T(1, in), 2), r_{25}: (s \to (s, in), 1),$
$r_{26}: (s \to g(u, out)|_f, 0), r_{27}: (0k \to 0(k, in), 4), r_{28}: (Tk \to T(k, in), 4),$
$r_{29}: (1k \to 1(k, in), 4), r_{30}: (u0 \to \delta u, 1), r_{31}: (u0k \to \delta u(k, in), 0),$
$r_{32}: (u \to x|_A, 0), r_{33}: (u \to x|_B, 0), r_{34}: (u \to x|_C, 0), r_{35}: (xk \to (ix, out), 1),$
$r_{36}: (x \to (x, out), 2), r_{37}: (0i \to 1(i, out)|_A, 1), r_{38}: (0i \to 0(i, out)|_B, 1),$
$r_{39}: (0i \to T(i, out)|_C, 1), r_{40}: (Ti \to 0(i, out)|_A, 1), r_{41}: (Ti \to T(i, out)|_B, 1),$
$r_{42}: (Ti \to (T, in)1(i, out)|_C, 1), r_{43}: (1i \to (1, in)T(i, out)|_A, 1),$
$r_{44}: (1i \to 1(i, out)|_B, 1), r_{45}: (1i \to 0(i, out)|_C, 1),$
$r_{46}: (v \to (v, out)|_{1A}, 1), r_{47}: (v \to (v, out)|_{0B}, 1), r_{48}: (v \to (v, out)|_{TC}, 1),$
$r_{49}: (v \to (v, out)|_{1B}, 1), r_{50}: (v \to (v, out)|_{0C}, 1), r_{51}: (v \to (r, out)|_{1C}, 1),$
$r_{52}: (v \to (r, out)|_{0A}, 1), r_{53}: (v \to (r, out)|_{TB}, 1), r_{54}: (v \to (r, out)|_{TA}, 1),$
$r_{55}: (r \to (r, out), 1), r_{56}: (V \to (v, out)|_f, 0), r_{57}: (V \to (V, in), 1),$
$r_{58}: (gk \to [1g], 0), r_{59}: (1g \to 1[g], 1), r_{60}: (fk \to f(1, in), 1)\}$

In this case, the correspondence of some substances is as follows. The dividends T, 0, 1 are represented as the objects a, b, c when they enter the membrane M_1 from the membrane 1. When they arrive in the membrane where f is located, and are represented as the objects T, 0, 1 after reacting with f. The divisors T, 0, 1 enter membrane M_1 as objects c, b, a, and arrive in the membrane where E is located. They are represented as objects A, B, C after reacting with E. Objects s, f, E, and Y function in the same way as addition. Objects K, M, N, H, e are used to control the generation of object k. Object u is used to determine whether the digits of the dividend are equal to the divisor.

Object u is converted to x when it encounters the divisor. Object k produced after the dividend and the divisor digits are the same, will be consumed with x and i produced. Object i converts the dividend to the state it was in when it was just the same number of digits as the divisor. Send V from membrane 1 into the innermost submembrane of membrane M_1. If the dividend is greater than the divisor, return v to membrane 1 to produce k, and continue with the addition; if it is less than that, return r to indicate that no more addition can be performed. At the end of the reaction, the quantity of k is the decimal representation of the quotient, and g converts the quantity of k to symmetric ternary.

In this paper, division of two numbers is actually accomplished by iterative subtraction, where the divisor is subtracted from the dividend. The cycle of subtraction continues until the dividend is less than the divisor. The number of rounds of subtraction is the quotient, and the remaining dividend that cannot be subtracted anymore is the remainder. In symmetric ternary, subtraction is converted to addition by simply changing the non-zero bit of the divisor to

its opposite, i.e., 1 to T and T to 1. So, we can change iterative subtraction to iterative addition. The following modules are used in the Division P System:

- **Module 1** (R_1: $r_1 \sim r_8$ and R_M: $r_1 \sim r_9$) is the input of dividend and divisor. The dividend is input and stored in each layer of the membrane as "T, 0, 1", and the non-zero bit of the divisor is turned into its opposite and stored in each layer of the membrane as "A, B, C".
- **Module 2** (R_M: $r_{10} \sim r_{24}$) is the iterative addition of the dividend and the opposite of the divisor. When the number of bits of the dividend is greater than the number of bits of the divisor, no judgment is required to perform the addition operation.
- **Module 3** (R_M: $r_{37} \sim r_{45}$) is a module that restores the dividend to the state when the number of digits of the dividend is the same as the number of digits of the divisor. And at the same time, object k is consumed in its entirety, and the generation of k is suppressed in Membrane 1.
- **Module 4** (R_M: $r_{46} \sim r_{57}$) is a module that determines whether the dividend is greater than the divisor. If the dividend is greater than the divisor, the addition continues; if it is less, the reaction stops.

Let us assume that the dividend is $a_n a_{n-1}...a_1$ and the divisor is $b_m b_{m-1}...b_1$ (where $n \geq m$), and we illustrate the use of the rules in Π' below by dividing two numbers.

(1) Input of the dividend:

First, a_1 is sent into membrane 1.

- **Case 1**: $a_1 = T$, rule r_1 in R_1 is applied, and object T is converted to a in the presence of Y and is sent into membrane M_1. Rule r_1 in R_M is executed, and a is converted into T while a new membrane M_2 is created, and f enters membrane M_2.
- **Case 2**: $a_1 = 0$, rule r_2 in R_1 is applied, and object 0 is converted to b in the presence of Y and is sent into membrane M_1. Rule r_2 in R_M is executed, and b is converted into 0 while a new membrane M_2 is created, and f enters membrane M_2.
- **Case 3**: $a_1 = 1$, rule r_3 in R_1 is applied, and object 1 is converted to c in the presence of Y and is sent into membrane M_1. Rule r_3 in R_M is executed, and c is converted into 1 while a new membrane M_2 is created, and f enters membrane M_2.

Object a_i ($1 < i < n$) is sent into membrane 1 and converted to object a, b, or c into membrane M_1. Then it continues into the inner membrane until it reaches membrane M_i where it is consumed with f and is converted to object T, 0 or 1. The last bit of the dividend a_n is sent in with object s to indicate that the dividend has been fully entered. When object s enters membrane 1, rule r_7 in R_1 is applied. Object E is produced, signaling the system that it is ready to enter the divisor, while s is sent into membrane M_1. After that, object s continues into the inner membrane (R_M: r_{25}) until it reaches membrane M_n. In membrane M_n, object s is converted to u and g (R_M: r_{26}) catalyzed by f. Object u is exported to membrane M_{n-1} (R_M: r_{26}), and g stays in membrane M_n.

(2) Input of the divisor:

Object b_1 is sent into membrane 1:

- **Case 1**: $b_1 = T$. Rule r_4 in R_1 is executed, and object T is converted to c in membrane M_1. Then, rule r_9 in R_M is applied, object c is converted to C, and E enters membrane M_2.
- **Case 2**: $b_1 = 0$. Rule r_5 in R_1 is executed, and object 0 is converted to b in membrane M_1. Then, rule r_8 in R_M is applied, object b is converted to B, and E enters membrane M_2.

- **Case 3**: $b_1 = 1$. Rule r_6 in R_1 is executed, and object 1 is converted to a in membrane M_1. Then, rule r_7 in R_M is applied, object a is converted to A, and E enters membrane M_2.

Object b_i ($2 \leq i \leq m$) is sent into membrane 1, then it is converted to object a, b, or c into membrane M_1. It continues into the inner membrane until it reaches membrane M_i where it is consumed with E to produce object A, B, or C. The highest bit of the divisor b_m is sent into membrane 1 with object s. When s is sent into membrane 1, rule r_9 in R_1 is applied, object s is consumed with K to produce E and K, which is used to produce object k.

(3) Division:

When the divisor is fully sent into the system, object k is produced in Membrane 1 (R_1: $r_9 \sim r_{10}$), and k is sent into membrane M_1 to trigger addition. Then one k is produced every three time slices into membrane M_1 (R_M: $r_{10} \sim r_{14}$).

The possibilities of $a_1 + b_1$ are as follows:

- **Case 1**: $0 + T = T$ (R_M: r_{10});
- **Case 2**: $0 + 0 = 0$ (R_M: r_{11});
- **Case 3**: $0 + 1 = 1$ (R_M: r_{12});
- **Case 4**: $T + T = T1$ (R_M: r_{13});
- **Case 5**: $T + 0 = 0$ (R_M: r_{14});
- **Case 6**: $T + 1 = 0$ (R_M: r_{15});
- **Case 7**: $1 + T = 0$ (R_M: r_{16});
- **Case 8**: $1 + 0 = 1$ (R_M: r_{17});
- **Case 9**: $1 + 1 = 1T$ (R_M: r_{18}).

For cases 4 and 9 that generate feeds, the high bit of the result is sent into the inner membrane and the low bit remains in the original membrane.

Taking "0+T = T" as an example, the rule R_M: r_{10} indicates that 0 is consumed with k in the catalysis of A, generating T to remain in membrane M_1 and k to enter membrane M_2. The divisor, serving as a catalyst, remains unchanged to ensure that the size of the dividend decreases while the divisor remains the same.

When the first round of addition is completed and the first object k reaches the membrane M_{n+1}, where objects g and f are also present. The rule r_{58} in R_M is executed to convert k to 1 and create a new membrane M_{n+2}, sending 1 and g into membrane M_{n+2}. Then, rule r_{59} in R_M creates a new membrane M_{n+3} and sends g into membrane M_{n+3} to prevent result overflow. Thereafter, object f converts the k of membrane M_{n+1} to 1 (R_M: r_{60}), sending it into membrane M_{n+2} and converting the quotient to a symmetric ternary number. Object g encountering object 1 creates a new membrane, preventing overflow.

As addition proceeds, the dividend $a_n a_{n-1} \ldots a_1$ decreases, and when $a_n = 0$, object u dissolves the membrane M_n (R_M: r_{30}). This continues until u dissolves the membrane M_{m+1}, and encounters the divisor (R_M: $r_{32} \sim r_{34}$). At this point, the bits of the dividend and the divisor are the same. Further additions should then check if the dividend is greater than the divisor, allowing addition to continue only if this condition is met. However, membrane 1 still produces one k every three time slices, which requires reversing the additions done after the numbers have the same number of digits and eliminating the k produced, sending a signal to stop k production in membrane 1.

When the digits of the dividend and divisor align, object u is converted to x. x is then transported out of the membrane (R_M: r_{36}) until it encounters k or reaches membrane 1. If x meets k, rule R_M: r_{35} is enacted to produce i and x. Object i reverses the addition performed for this k (R_M: $r_{37} \sim r_{45}$), and x continues to membrane 1. Upon reaching, rule R_1: r_{15} is executed, generating X and V. X regulates k production, while V moves to membrane M_{m+1}, converting to v catalyzed by f (R_M: r_{56}). v assesses whether the dividend exceeds the divisor, returning to membrane 1 if greater, or sending r to indicate that no further addition can be done (R_M: $r_{46} \sim r_{54}$).

When v returns to membrane 1, rule r_{16} in R_1 is executed, producing k and sending it to membrane M_1 for further addition, while V continues into membrane M_{m+1}, converted to v by f. The process iterates until r returns to membrane 1, signaling the cessation of reactions.

The remainder of the division is stored in membranes $M_1, \ldots, M_i$, and the quotient in membranes $M_{i+1}, \ldots, M_f$ ($f > i + 1$). The division example in Section 4.1.3 allows a more concrete demonstration of the implementation of the rule. In the division P System Π', the symmetric ternary numbers of n/m ($n \geq m$) bits require the following time slices to implement the division operation: The time slices required by the input module is $3n + 3m - 2$, The time slices required by the iterative addition module is $4i$, The time slices required by the revert module is $2m$, The time slices required by the evaluation module is $(2m + 3)j$, where $i + j$ = quotient, i is the quotient that results when the number of dividend digits is greater than the divisor, and j is the quotient that results when the number of dividend digits is equal to the divisor. The efficiency of the system will be greatly improved when the number of dividend digits is much larger than the divisor.

3.4 Comparison of computational efficiency

In this section, we analyse and compare P systems proposed in recent years for basic arithmetic operations, the results are shown in Table 5. The statistics include the number of rule types used for the four basic operations (addition, subtraction, multiplication, and division), and the number of time slices required to complete the operations. In [13, 14], m and n respectively represent the two decimal numbers used in the operation, and n = max {m, n}. In this paper, m and n respectively represent the two symmetric ternary numbers used in the operation, and n = max {m, n}.

In Table 5, the arithmetic P systems designed in [13–16] are all decimal based. And instead of considering the input of data, the numbers to be computed are directly put into the membranes. In other words, they did not take into account the input of computational data. [13] designed an arithmetic P system based on a multi-layer membrane, and [14] designed an arithmetic P system based on a single membrane In [15] the time slices of the operations is not given, only it is mentioned that the complexity of these operations in P system is "linear".

Table 5. Time slices required and number of rule types used for four arithmetic approaches.

Article	Rule type	Add	Sub	Mul	Div
[13]	5/5/6/11	O(n)	O(n)	O(m)	O(n)
[14]	2/1/11/12	O(1)	O(1)	O(max (n,m))	linear
[15]	39/39/29/34	linear	linear	linear	linear
[16]	3 or 4/4/11/10	-	-	-	-
This Work	21/21/43/78	O(4m)	O(4m)	O(4m)	O(3n)

Note: The 'Rule type' column is represented in the form of A/B/C/D, where A represents the number of rules used for addition. Similarly, B, C, and D denote the number of rule types used for subtraction, multiplication, and division, respectively.

https://doi.org/10.1371/journal.pone.0312778.t005

Since [16] discusses arithmetic operation and arithmetic expression evaluation in transition P system based on rules with priority, no arithmetic operations are performed. Therefore, the time slices required for arithmetic operations is marked with '-'.

From the above analysis, it can be seen that this paper not only considers the input of the data, the dynamic generation and dissolution of the membrane, but also introduces the symmetric ternary number system, which eliminates the influence of the operational sign on the calculation. The arithmetic operation P system designed in this paper is more novel and has a wider application scenario.

4 Simulation and validation of rules

This section is based on Section 3 to further validate the correctness of the rules in the P System. The rules in Section 3 are simulated and experimented with UPSimulator [27]. The main discussion is the simulation experiments of (1) addition: $1T00 + 1T1 = 10T1$; (2) multiplication: $1TT \times 1T = 101$; and (3) division: $101/1T = 1TT$.

4.1 Examples of symmetric ternary arithmetic operations

4.1.1 Example of addition. Here is an example of "$1T00 + 1T1$" to illustrate the implementation of the addition rules.

The initial state of Π^+ is shown in Fig 2. Firstly, objects 0, 0, T, 1 are sequentially sent into membrane 1 every two time slices. When object 0 is input to membrane 1, it is converted to object b and is sent into membrane M_1 immediately. Object b is consumed with f in membrane M_1 and object 0 produced, at the same time, a new membrane M_2 is created with f entering membrane M_2. The next input of object 0 is converted to object b and is sent into membrane M_2, which then is consumed with f and object 0 produced, and creates a new membrane M_3, with f entering membrane M_3. Object T is input to membrane 1, it is converted to object a and is sent into membrane M_3. Object a is consumed with f and object T produced, and creating a new membrane M_4, with f entering membrane M_4.

When the highest bit object 1 is input, object s is input at the same time to indicate the end of the augend number. Objects 1 and s are input to membrane 1, and object 1 is converted to object c which is sent into membrane M_4. Object c is consumed with f and object 1 produced, and creates a new membrane M_5, with f entering into membrane M_5. Object s is consumed with Y in membrane 1 and E produced, which indicates that the addend number can be input

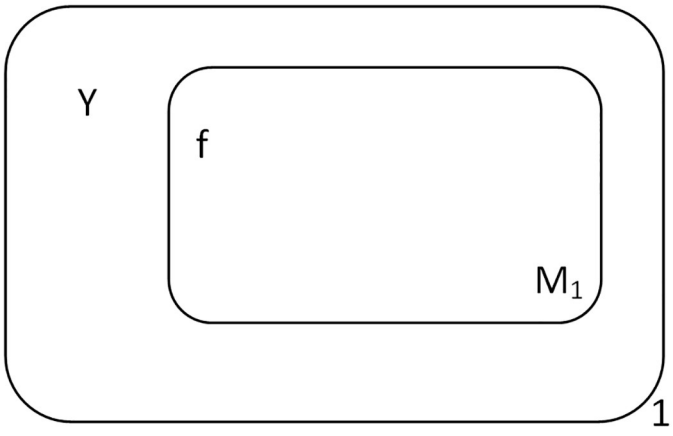

Fig 2. Initial configuration of Π^+.

https://doi.org/10.1371/journal.pone.0312778.g002

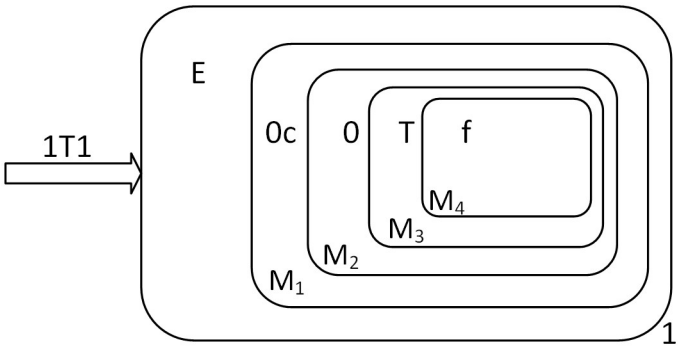

Fig 3. The configuration of addend waiting for input.

https://doi.org/10.1371/journal.pone.0312778.g003

now, and at the same time, object E enters into membrane M_1. Fig 3 shows the membrane structure where the augend is input completely and the addend waits to be input.

The lowest bit of the addend, object 1, is sent into membrane 1. Object 1 is converted to c and is sent into membrane M_1. Object c and E are consumed with object 1 produced, and object E is sent into membrane M_2. Object 1 is added to the 0 in membrane M_1, and the generated results is 1, which is stored in membrane M_1.

Input the second bit of the addend, object T, to membrane 1. Object T is converted to a, which is sent into membrane M_2, where it is consumed with E to generate object T. Object T and the object 0 in membrane M_2 are consumed with object T produced, which is stored in membrane M_2.

Send the last bit of the addend, object 1, to membrane 1. Object 1 is converted to c, which is sent into membrane M_3 and reacts with E to generate object 1. Object 1 is consumed with object T in membrane M_3 and object 0 produced, which is stored in membrane M_3.

At this point, there are no more rules in the system that can be executed, the system stops, and the obtained result "10T1" is saved in low to high order in $M_{1,\ldots,4}$ as shown in Fig 4. The rules of the entire system run in parallel. While the augend is still moving towards the inner membrane and building new membranes, the addend has already been input to start the addition operation. In this way, the whole system can perform the operation quickly.

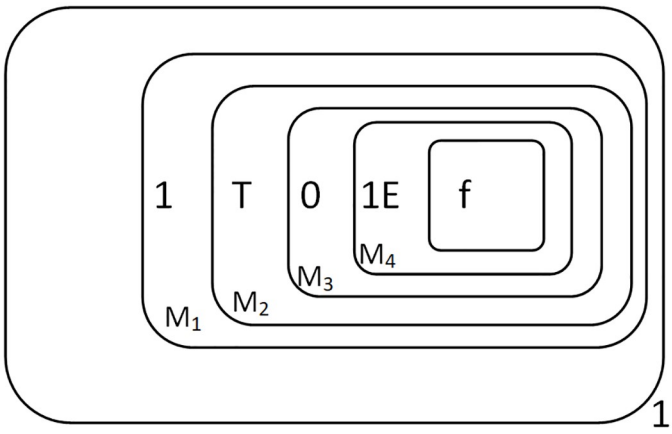

Fig 4. The configuration at completion of addition.

https://doi.org/10.1371/journal.pone.0312778.g004

Tables 6 and 7 show the process of object changes in each membrane during the whole system run. The number of digits of the augend and the addend also has an effect on the time slices. For example, in Table 6, it takes 23 time slices for the augend to be 1*T*00 (four-digit number) and the addend to be 1*T*1 (three-digit number). While in Table 7, it takes 24 time slices for the augend to be 1*T*1 (three-digit number) and the addend to be 1*T*00 (four-digit number). Therefore, when using this arithmetic system, the one with more digits can be selected as the augend to reduce the time slices.

4.1.2 Example of multiplication. Here is an example of "1*TT* × 1*T*" to illustrate the implementation of the multiplication rules.

The input of the multiplicand is the same as the augend, so we won't go into too much detail here. The multiplicand "1*TT*" is finally stored in the form of "*AAC*" in the membranes M_1, M_2, M_3. Object *s* is consumed with Object *Y* in membrane 1 and object *E* produced, at which point the multiplier object *T* can be input, and *E* then enters the membrane M_1. The state of the membrane system when the multiplier is about to be input is shown in Fig 5.

The lowest bit of the multiplier, object *T*, is input. Object *T* is consumed with *n* to generate *u*, and *T* is converted to object *a* into membrane 1. Object *a* is consumed with *E* to produce *x* while *E* enters the membrane M_2. Object *x* is consumed with *A* to produce object 1 which is retained in membrane M_1, and *A* is converted to object *p* which enters membrane M_2 with *x*. In membrane M_2, object *x* is consumed with *A* to produce object 1 which is retained in membrane M_2. Object *A* is converted to *p* which enters membrane M_3 with *x*. In membrane M_3, object *x* is consumed with *C* to produce *T* to be retained in membrane M_3, and *C* is converted

Table 6. Process of object changes in each membrane during the addition. (1T00+1T1).

Time Slice	Membrane 1	M1	M2	M3	M4	M5
0	Y	f				
1	0Y	f				
2	Y	bf				
3	Y	0	f			
4	0Y	0	f			
5	T	0b	f			
6	Y	0	bf			
7	TY	0	0	f		
8	Y	0a	0	f		
9	Y	0	0a	f		
10	1sY	0	0	af		
11	E	0c	0	T	f	
12	1	0E	0C	T	f	
13		0cE	0	cT	f	
14		01	0E	T	1	f
15	T	1a	0E	T	1	f
16		1	0E	T	1	f
17		1	0Ea	T	1	f
18	1	1c	0T	TE	1	f
19		1	T	TE	1	f
20		1	cT	TE	1	f
21		1	T	TEc	1	f
22		1	T	T1	1E	f
23		1	T	0	1E	f

https://doi.org/10.1371/journal.pone.0312778.t006

Table 7. Process of object changes in each membrane during the addition. (1T1+1T00).

Time Slice	Membrane 1	M1	M2	M3	M4	M5	M6
0	Y	f					
1	1Y	f					
2	Y	cf					
3	Y	1	f				
4	TY	1	f				
5	Y	1a	f				
6	Y	1	af				
7	1sY	1	T	f			
8	E	1c	T	f			
9	0	1E	Tc	f			
10		1Eb	T	cf			
11		10	TE	1	f		
12	0	1	TE	1	f		
13		1b	TE	1	f		
14		1	TEb	1	f		
15	T	1	T0	1E	f		
16		1a	T	1E	f		
17		1	Ta	1E	f		
18	1	1	T	1Ea	f		
19		1c	T	1T	Ef		
20		1	cT	0	E	f	
21		1	T	0c	E	f	
22		1	T	0	Ec	f	
23		1	T	0	1	Ef	
24		1	T	0	1	E	f

https://doi.org/10.1371/journal.pone.0312778.t007

to r which is sent into membrane M_4 with x. Object x and f in the membrane M_4 are consumed to create a new membrane M_5, while f is sent into membrane M_5 to prevent the result from overflowing.

At this point, the objects "CCA" are converted to "ppr" and moved one layer into the membrane. In fact, when object T is sent into the membrane system to carry out the above biochemical reaction, object 1 is also sent into the membrane to start the calculation. Fig 6 shows

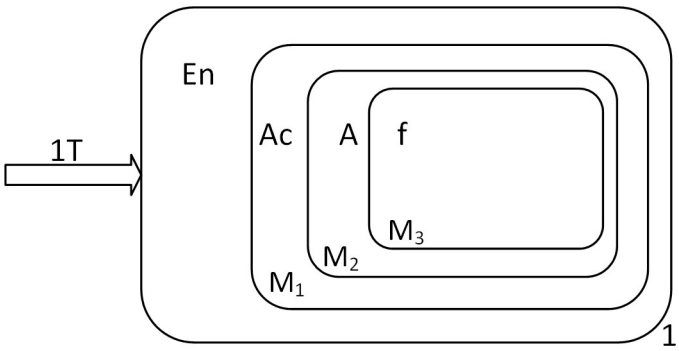

Fig 5. The configuration of Π* when the multiplier is about to be input.

https://doi.org/10.1371/journal.pone.0312778.g005

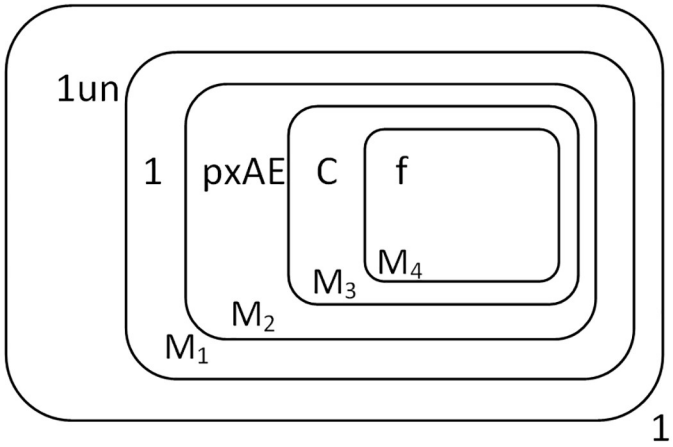

Fig 6. The configuration of Π* when the multiplier is input completely.

https://doi.org/10.1371/journal.pone.0312778.g006

the state of the membrane system when object 1 has just been sent into the membrane, and object T has just arrived in the membrane M_2 about to undergo the next calculation, and these calculations are in parallel.

When object 1 is input, object u will be sent into membrane M_1 and reach membrane M_2, which converts p to A. It continues to move to the inner membranes, converting p and r to A and C. Object 1 is converted to c to be sent into membrane M_2, which is consumed with E to generate z. Object z is consumed with A to produce T, and Object A is converted to p, which is sent into membrane M_3 with z. Object T is consumed with the previously generated object 1 and object 0 produced. In membrane M_3, object z and A are consumed to produce T, and then A is converted to p along with z into membrane M_4. Object T in membrane M_3 is consumed with the previously generated T to produce objects $T1$, with 1 retained in membrane M_3 and T as a feed into membrane M_4. In membrane M_4, object z is consumed with C to produce 1, and then C is converted to r along with z into membrane M_5. Object 1 in membrane M_4 is consumed with the previously fed T to generate 0 to be retained in membrane M_4. Object x is consumed with f to create a new membrane M_6 while f is sent into membrane M_6.

At this point, the reaction was completed. The result "1010" is preserved in the membranes M_1 to M_4 as shown in Fig 7.

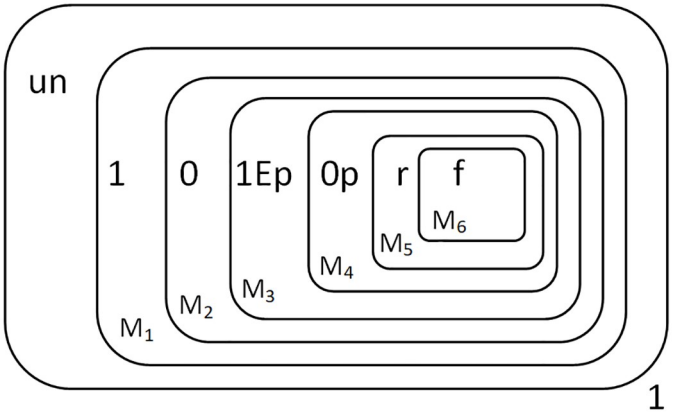

Fig 7. The configuration at the completion of multiplication.

https://doi.org/10.1371/journal.pone.0312778.g007

Table 8. Process of object changes in each membrane during the multiplication.

Time Slice	Membrane 1	M1	M2	M3	M4	M5	M6
0	Y	f					
1	TY	f					
2	Y	af					
3	Y	A	f				
4	TY	A	f				
5	Y	Aa	f				
6	Y	A	af				
7	1sY	A	A	f			
8	En	Ac	A	f			
9	Tn	AE	Ac	f			
10	un	AEa	A	cf			
11	un	Ax	AE	C	f		
12	1un	1	AEpx	C	f		
13	1n	1u	1pE	Cpx	f		
14	un	1c	1pEu	Tp	rxf		
15	un	1	1AEc	Tpu	r	f	
16	un	1	1Az	TAE	ru	f	
17	un	1	1T	TAEpz	C	uf	
18	un	1	0	TTEp	Cpz	f	
19	un	1	0	1Ep	T1p	rzf	
20	un	1	0	1Ep	0p	f	f

https://doi.org/10.1371/journal.pone.0312778.t008

Table 8 shows the process of object changes in each membrane as the entire system runs, taking a total of 20 time slices.

4.1.3 Example of division. Here is an example of "101/1*T*" to illustrate the implementation of the division rules.

The input of the divisor is the same as the augend. When the highest bit of the dividend, object 1, is sent into membrane 1 with object *s*, *s* is consumed with *Y* to produce *E*. At the same time, *s* will enter the membrane M_1 until it reaches the membrane M_4. Object *s* is converted to *u* and *g* catalyzed by *f*. Object *u* is transported to membrane M_3, where the highest bit of the dividend is located. Object *g* remains in membrane M_4. When *E* is detected in the system, the divisor can be entered. First input *T*, Object *T* is converted to *c* into membrane M_1. Object *c* and *E* are consumed to generate *C*, and *E* is sent to membrane M_2. Input 1 and *s*, Object 1 is converted to object *a* into membrane M_2. Object *a* and *E* are consumed to generate *A*, and *E* is sent to membrane M_3. Object *s* and *K* are consumed to generate object *e* and *K*. Object *e* and *K* execute the rules r_{10} to r_{14} in R_1, and one *k* is produced every four time slices. Then *k* is sent into membrane M_1 to trigger addition operation. The state of the membrane system when all the divisors are sent into membrane M_1 is shown in Fig 8.

The first *k* enters membrane M_1, object 1 and *k* are consumed to generate 1 and *T* in the presence of *C*, then 1 enters membrane M_2 as a feed along with *k*. Object *k* then triggers addition in membrane M_2, and so on, and finally will reach membrane M_4. Object *k* and *g* in membrane M_4 are consumed to create a new membrane M_5, and object 1 and *g* enters membrane M_5. And then 1 and *g* are consumed to create a new membrane M_6 and *g* is sent into membrane M_6.

After three rounds of addition, the highest digit of the dividend turns to 0. Object *u* detects 0, dissolves the current membrane, and the objects in membrane M_3 fall into membrane M_2.

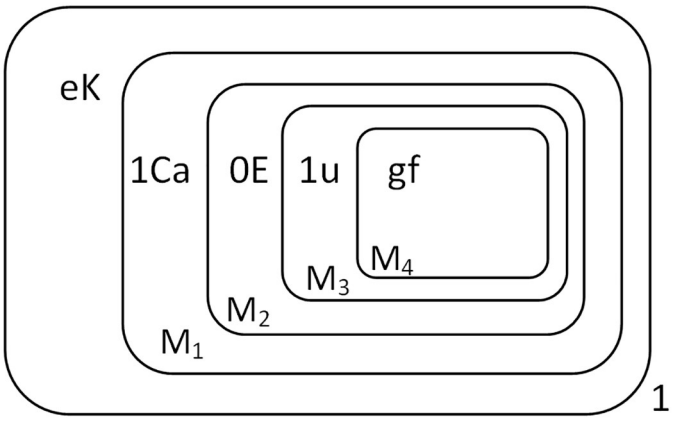

Fig 8. The configuration when all divisors are input to membrane M_1.

https://doi.org/10.1371/journal.pone.0312778.g008

Object u detects the highest digit of the divisor in membrane M_2, indicating that the digits of the dividend and the divisor are now the same. At this point it is not possible to directly perform the addition operation, but to compare the dividend and the divisor before deciding whether it is possible to perform the addition. However, the system continues to produce the object k, and the configuration of the system when the fourth K is generated is shown in Fig 9.

First, we have to restore the addition operation after the third k, and send a signal to membrane 1 to stop producing k. Object u is converted to x catalyzed by A, and k will be consumed to avoid another addition operation. At the same time, object i will be produced, which will restore the dividend to the state where it did three addition operations (R_M:$r_{37} \sim r_{45}$). Eventually object i and x will be transported to membrane 1, where x is consumed with e to prevent the generation of k, and produce X and V. Object V is sent into membrane M_4, object V is consumed to produce v catalyzed by f. Object v is sent to membrane M_3 to compare the dividend and the divisor. If the dividend is greater than the divisor, then X produces k to continue the addition.

In this example, the first round of testing determines that the dividend is greater than the divisor, so object v is sent to membrane 1. Rule r_{16} in membrane 1 is executed, producing k which is sent into membrane M_1 to perform addition. At the same time V is produced to prepare for the second round of testing. The second round of testing detects that the dividend is

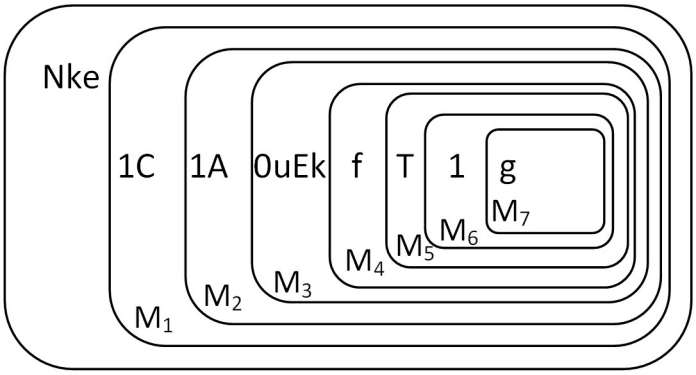

Fig 9. The configuration when the fourth k is just generated.

https://doi.org/10.1371/journal.pone.0312778.g009

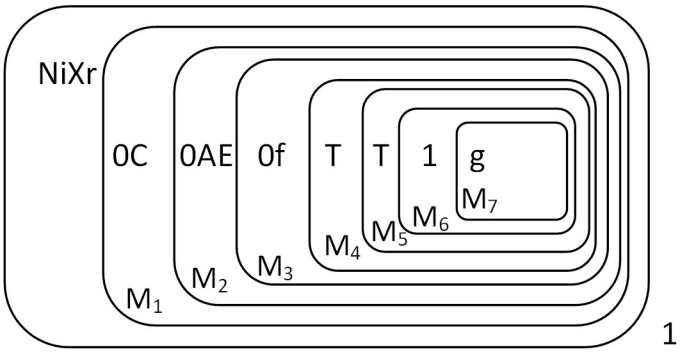

Fig 10. The configuration at the completion of division.

https://doi.org/10.1371/journal.pone.0312778.g010

still greater than the divisor and sends v to membrane 1, performing the same steps as above. The third round of testing judges that no more addition can be performed, returning r to membrane 1.

At this point, there are no more rules in the system to execute and the system stops. The result of the quotient is saved in membranes $M_{4,\ldots,6}$, and if there is a remainder, it is saved in membranes $M_{1,\ldots,3}$. In this example, there is no remainder. The quotient "1TT" is preserved in the membranes $M_{4,\ldots,6}$ as shown in Fig 10.

Table 9 shows the process of object changes in each membrane as the division system runs. Due to space constraints, only a portion of the table is shown. It takes a total of 51 time slices.

4.2 Simulation of addition

For the simulation of Π^+, the rules in Π^+ are described in UPLanguage [27]. The membrane class "M" (i.e., membrane 1) is defined to contain the membrane class "Add", membrane class "B" and membrane class "C". Membrane classes B and C are used to input the augend and addend respectively. The overall membrane structure is as follows:

```
Environment {
    Membrane M m1 {
        Object Y;
        Membrane Add A1 {
            Object f;
        }
        Membrane B B1 {
        Object O, B, T, I, s;
        }
        Membrane C C1 {
        Object C, T, I;
        }
    }
}
```

Where 'Object' is used to specify the objects used in the simulation. The above rules indicate that initially the membrane $m1$ contains one object Y, the membrane A_1 contains one object f, the membrane B_1 contains objects O, B, T, I, s, and the membrane C_1 contains objects

Table 9. Process of object changes in each membrane during the division.

Time Slice	Membrane 1	M1	M2	M3	M4	M5	M6	M7	M8
. . .	. . .	. . .							
12	1sK	1C	0E	1	sf				
13	eK	1Ca	0E	1u	gf				
14	Nke	1C	0Ea	1u	gf				
15	Me	1Ck	0A	1uE	gf				
16	He	TC	0A1k	1uE	gf				
17	Ke	TC	1Ak	1uE	gf				
18	Nke	TC	0A	1uEk	gf				
19	Me	TCk	0A	1uE	gfk				
20	He	0C	0Ak	1uE	f	1g			
21	Ke	0C	TA	1uEk	f	1	g		
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .		
26	Nke	1C	1A	0uEk	f	T	1	g	
27	Me	1Ck	1AuE		fk	T	1	g	
28	He	TC	1k 1AxE		f	T	1	g	
29	Ke	TCix	TAE		1f	0	1	g	
30	Nke ix	1C	TTAE		1f	0	1	g	
31	NkiX	1CV	1AE		1Tf	0	1	g	
32	NiX	1C	1AEV		0f	0	1	g	
33	NiX	1C	1AE		V0k	0	1	g	
34	NiX	1C	1AEv		0f	0	1	g	
35	NiX	1Cv	1AE		0f	0	1	g	
36	NiXv	1C	1AE		0f	0	1	g	
37	NiXV	1Ck	1AE		0f	0	1	g	
38	NiX	TCV	1k 1AE		0f	0	1	g	
. . .	. . .	. . .	. . .		. . .	. . .	. . .	. . .	
51	NiXr	0C	0AE		0f	T	T	1	g

https://doi.org/10.1371/journal.pone.0312778.t009

C, *T*, *I*. It should be noted that the UPS can't use numbers to represent the objects, so we use object *I* instead of 1, object *O* instead of 0. When the system starts running, it will output objects *O*, *O*, *T*, and *Is* into membrane m_1 every two time slices from membrane *B*. Object *s* and *Y* are converted to *E* and enters all submembranes. When Object *E* enters membrane *C*, Membrane *C* begins to output addend numbers.

Here is a brief description of the format in which the rules are written in UPS.

- **Rule** r_1: $T \rightarrow$ (*a*, in all), 1; this means that object *T* is converted to object *a* and enters all submembranes with priority 1.
- **Rule** r_2: $af \rightarrow T$Add:$a\{f\}$, 1; this denotes that object *a* and *f* are converted to object *T* and creates a new membrane that inherits the rules of the Add membrane class and that object *f* goes into the newly created membrane.

Rewriting the rules in Section 3.1 in the above format, the result is shown in Fig 11. Objects *I*, *T*, *O*, *I* are retained in membranes $M_{1,\ldots,4}$ separately, and a total of 24 time slices are used. It is one time slice more than the actual projection, because object *E* has to enter membrane *C* before releasing the addition in UPS, this will consume one time slice.

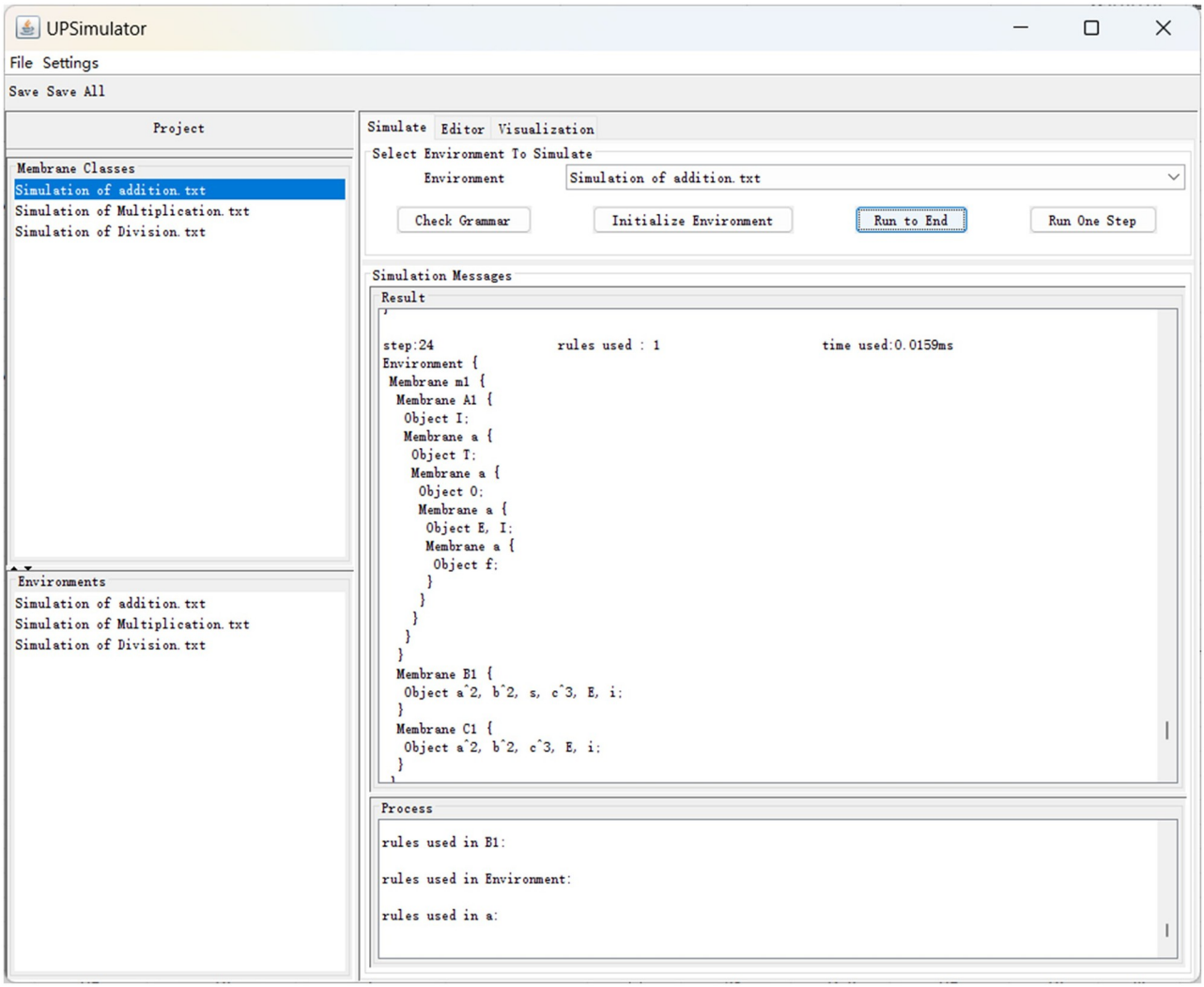

Fig 11. Simulation of *1T00+1T1 = 10T1*.

https://doi.org/10.1371/journal.pone.0312778.g011

4.3 Simulation of multiplication

The membrane structure for multiplication is the same as for addition; the membrane class "M" contains the membrane class "Mul", membrane class "B" and membrane class "C". Membrane classes B and C are used to input the multiplicand and multiplier into membrane m_1. The overall membrane structure is as follows:

```
Environment {
    Membrane M m1 {
        Object Y;
        Membrane Mul A1 {
            Object f;
        }
        Membrane B B1 {
            Object A, T, I, s;
```

```
        }
        Membrane C C1 {
            Object T, I;
        }
    }
}
```

Rewriting the rules in Section 3.2 with UPLanguage and running it yields the results as shown in Fig 12. Objects I, O, and I are retained in membranes $M_{1,\ldots,3}$. It takes a total of 21 time slices. This duration is one time slice more than the projection because, in the UPS, object E must enter membrane C before releasing the multiplier.

4.4 Simulation of division

For the simulation of Π', the rules of Π' are described in UPLanguage. We define a membrane class "M" which contains the membrane class "Div", membrane class "B" and membrane class

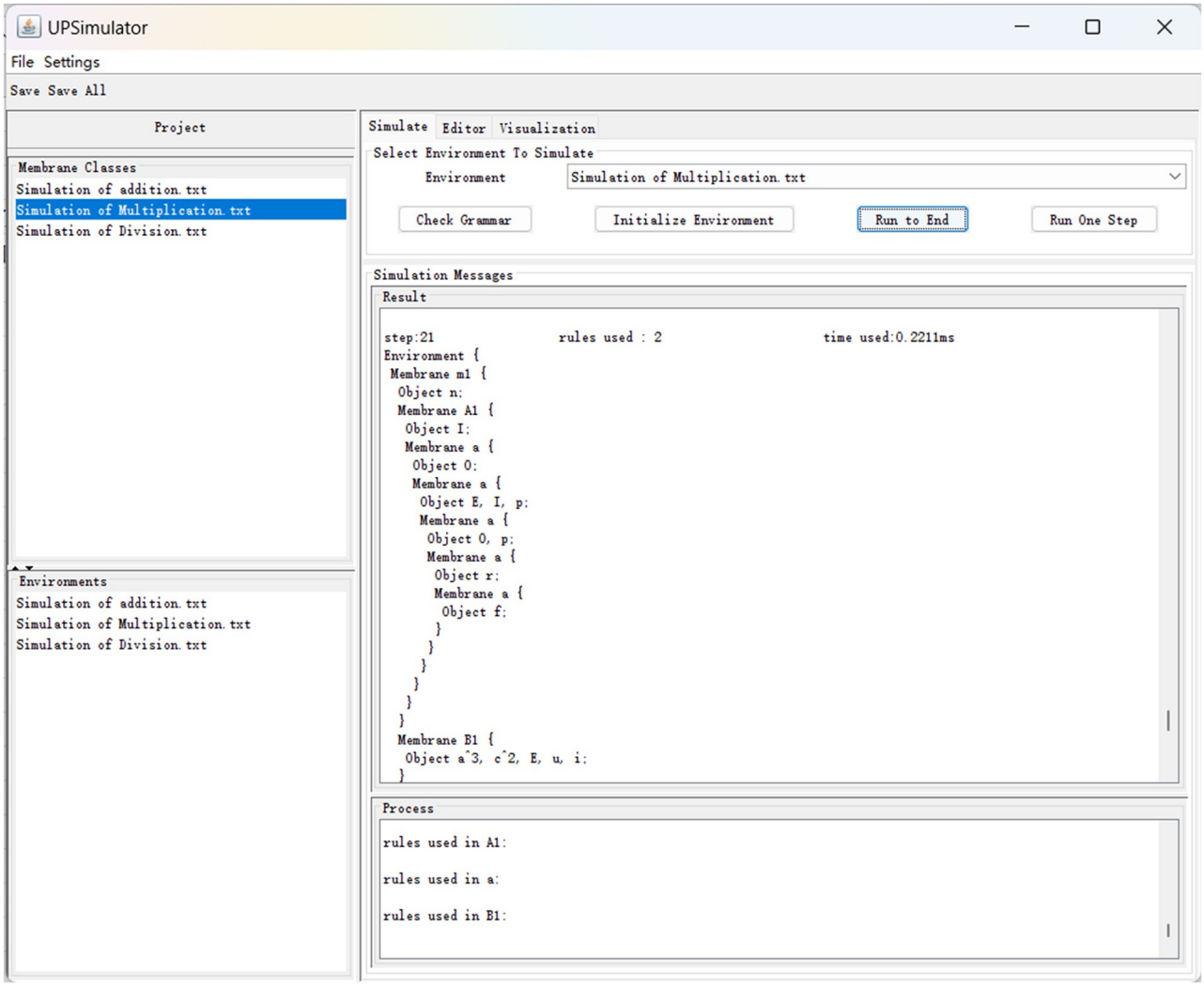

Fig 12. Simulation of *1TT*1T = 101*.

https://doi.org/10.1371/journal.pone.0312778.g012

"C". Membrane classes B and C are used to input the dividend and divisor into membrane m_1. The overall membrane structure is as follows:

```
Environment {
    Membrane M m1 {
        Object Y, K;
        Membrane Div A1 {
            Object f;
        }
        Membrane B B1 {
            Object I, O, C, s;
        }
        Membrane C C1 {
            Object T, I;
        }
    }
}
```

Division includes the operation of dissolving membranes, and the rules are reformulated in UPLanguage as follows:

- **Rule** r_1: $Ou \rightarrow$ dissolve(u, *out*), 1; this rule specifies that when objects O and u are present together, the membrane is dissolved.
- **Rule** r_2: $v \rightarrow (v, \text{out})|@O \,\&\, @C$, 1; this rule indicates that object v moves out only in the presence of both objects O and C.

Rewriting the rules in Section 3.3 with UPLanguage and executing the simulation yields the results depicted in Fig 13. The sequence *ITT* is retained in membranes M_5, M_6, M_7 in descending order, consuming a total of 54 time slices. This duration is three time slices longer than the initial projection because Object E has to enter membrane C before the divisor can be released, taking one additional time slice. Two extra time slices are consumed because, after the dividend and divisor are entered, object s is then introduced into membrane 1.

5 Conclusion

Membrane computing is characterized by parallelism, distribution, and uncertainty. It has been proved that membrane computing has equivalent computational capabilities with Turing machines, and its powerful parallel computing capability can effectively solve the bottleneck currently faced by electronic computers. The study of arithmetic operation system based on membrane computing has very important academic and practical significance for the realization of a general-purpose bio-computer.

In this paper, a symmetric ternary system is innovatively introduced, which is more adaptable in future bio-computers and can be closer to the natural computation of the human brain than the traditional binary system. A dynamic membrane structure based on membrane computing is designed, which makes the parallel operation of multi-digit numbers possible and improves the computational efficiency. Simulation results show that the designed P-system is not only suitable for basic arithmetic operations, but also can be extended to more complex computational tasks, which provides a new direction for the development of future computing devices.

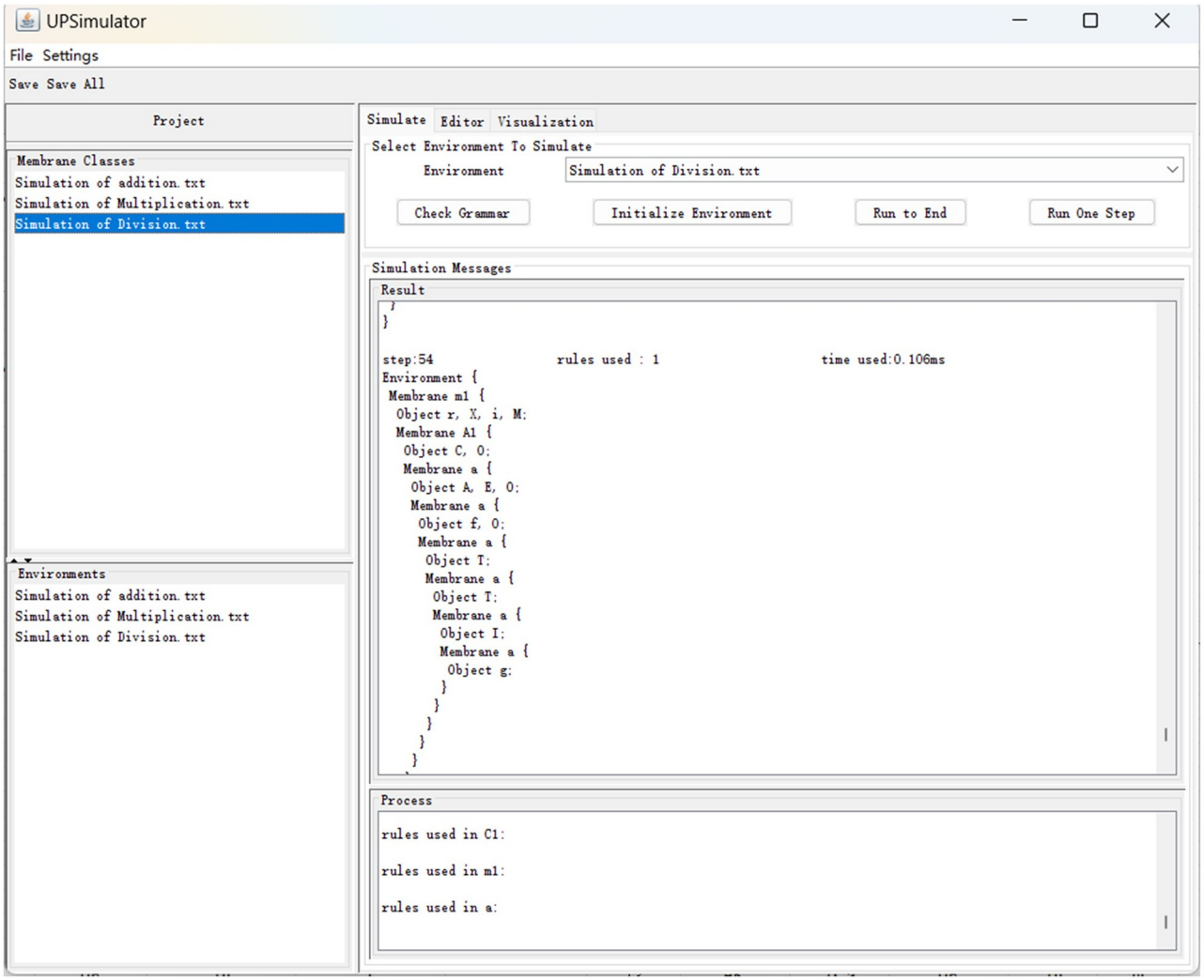

Fig 13. Simulation of *101/1T = 1TT*.

https://doi.org/10.1371/journal.pone.0312778.g013

In the P System we designed, 21 rules are used to implement addition and subtraction, symmetric ternary numbers "$n + m$" requires at most $3n + 4m$ time slices for addition. 43 rules are used to implement multiplication, "$n * m$" require at most $3n + 4m + 3$ time slices for multiplication. And 78 rules are used to implement division, n/m ($n \geq m$) require at most $3n + 5m - 2 + 4i + (2m + 3)j$ time slices, ($i + j$ = quotient, i is the quotient that results when the number of dividend digits is greater than the divisor, and j is the quotient that results when the number of dividend digits is equal to the divisor).

Future work will focus on further optimizing the performance of the system and exploring its applicability in more practical application scenarios. Also, we will further investigate their general structure to make it more versatile.

Author Contributions

Conceptualization: Hai Nan, Ping Guo.

Methodology: Jie Zhang.

Software: Jie Zhang, Jiqiao Jiang.

Supervision: Ping Guo.

Writing – original draft: Jie Zhang, Ping Guo.

Writing – review & editing: Hai Nan, Ping Guo, Xu Zhang.

References

1. Bessmertny I, Sukhikh N, Vedernikov J, Koroleva J. Ternary Logics in Decision Making. In: International Conference on Reliability and Statistics in Transportation and Communication. Springer; 2020. p. 411–419. https://doi.org/10.1007/978-3-030-68476-1_38
2. Alexander W. The ternary computer. Electronics and Power. 1964; 10(2):36–39. https://doi.org/10.1049/ep.1964.0037
3. Jin Y, He H, Lü Y. Ternary optical computer principle. Science in China Series F: Information Sciences. 2003; 46:145–150. https://doi.org/10.1360/03yf9012
4. Hu XM, Zhang C, Liu BH, Cai Y, Ye XJ, Guo Y, et al. Experimental high-dimensional quantum teleportation. Physical Review Letters. 2020; 125(23):230501. https://doi.org/10.1103/physrevlett.125.230501 PMID: 33337185
5. Paun G. Membrane computing: an introduction. Springer Science & Business Media; 2012.
6. Păun G. Computing with membranes. Journal of Computer and System Sciences. 2000; 61(1):108–143. https://doi.org/10.1006/jcss.1999.1693
7. Ciobanu G, Păun G, Pérez-Jiménez MJ. Applications of membrane computing. vol. 17. Springer; 2006.
8. Frisco P, Gheorghe M, Pérez-Jiménez MJ. Applications of membrane computing in systems and synthetic biology. Springer; 2014.
9. Buiu C, Vasile C, Arsene O. Development of membrane controllers for mobile robots. Information Sciences. 2012; 187:33–51. https://doi.org/10.1016/j.ins.2011.10.007
10. Păun G, Păun R. Membrane computing and economics: Numerical P systems. Fundamenta Informaticae. 2006; 73(1-2):213–227.
11. Atanasiu A. Arithmetic with membranes. Romanian Journal of Information Science and Technology, 2001, 4(1): 5–20.
12. Ciobanu G. A Programming perspective of the membrane systems. INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL. 2006; 1(3):13–24. https://doi.org/10.15837/ijccc.2006.3.2291
13. Guo P, Chen J. Arithmetic operation in membrane system. In: 2008 International Conference on BioMedical Engineering and Informatics. vol. 1. IEEE; 2008. p. 231–234. https://doi.org/10.1109/bmei.2008.136
14. Guo P, Zhang H. Arithmetic operation in single membrane. In: 2008 International Conference on Computer Science and Software Engineering. vol. 3. IEEE; 2008. p. 532–535. https://doi.org/10.1109/csse.2008.1212
15. Guo P, Luo M. Signed numbers arithmetic operation in multi-membrane. In: 2009 First International Conference on Information Science and Engineering. IEEE; 2009. p. 393–396. https://doi.org/10.1109/icise.2009.1062
16. Guo P, Liu SJ. Arithmetic expression evaluation in membrane computing with priority. Advanced Materials Research. 2011; 225:1115–1119. https://doi.org/10.4028/www.scientific.net/amr.225-226.1115
17. Guo P, Chen HZ. Arithmetic expression evaluation by P systems. Appl Math. 2013; 7(2L):549–553. https://doi.org/10.12785/amis/072l26
18. GUO P, CHEN H, ZHENG H. Arithmetic expression evaluations with membranes. Chinese Journal of Electronics. 2014; 23(1):55–60.
19. GUO P, ZHANG H, CHEN H, CHEN J. Fraction arithmetic operations performed by P systems. Chinese Journal of Electronics. 2013; 22(4):690–694.
20. Nan H, Kong Y, Zhan J, Zhou M, Bai L. P System with Fractional Reduction. Applied Sciences. 2023; 13(14):8514. https://doi.org/10.3390/app13148514

21. Liu J, Qi J, Yao J, Hu W, Zhang D, Xu HX, et al. Balanced-ternary-inspired reconfigurable vortex beams using cascaded metasurfaces. Nanophotonics. 2022; 11(10):2369–2379. https://doi.org/10.1515/nanoph-2022-0066

22. Faghih E, Taheri M, Navi K, Bagherzadeh N. Efficient realization of quantum balanced ternary reversible multiplier building blocks: A great step towards sustainable computing. Sustainable Computing: Informatics and Systems. 2023; 40:100908. https://doi.org/10.1016/j.suscom.2023.100908

23. Ratan Kumar S, Koteswara Rao L, Kiran Kumar M. Design of Ternary Multiplier Using Pseudo NCNTFETs. Russian Microelectronics. 2023; 52(2):119–127. https://doi.org/10.1134/s1063739723700245

24. Yunfu S, Zhehe W. Design and implementation of R4-MSD square root algorithm in ternary optical computer. Soft Computing. 2024; p. 1–14. https://doi.org/10.1007/s00500-023-09518-6

25. Malik A, Hussain MS, Hasan M. Energy-Efficient Exact and Approximate CNTFET-Based Ternary Full Adders. Circuits, Systems, and Signal Processing. 2024; 43(5):2982–3003. https://doi.org/10.1007/s00034-023-02589-8

26. Vudadha C. Design of CNFET-based Ternary Conditional Sum Adders using Binary Carry Propagation. In: 2024 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE; 2024. p. 1–5. https://doi.org/10.1109/iscas58744.2024.10558451

27. Guo P, Quan C, Ye L. UPSimulator: A general P system simulator. Knowledge-Based Systems. 2019; 170:20–25. https://doi.org/10.1016/j.knosys.2019.01.013

28. Păun G, Rozenberg G. A guide to membrane computing. Theoretical Computer Science. 2002; 287(1):73–100. https://doi.org/10.1016/s0304-3975(02)00136-6

29. Chen Q. T01 Ternary numeral system. Journal of Mathematics(China). 1958; 03:4–7.