Hindawi
Complexity
Volume 2021, Article ID 6817636, 13 pages
https://doi.org/10.1155/2021/6817636

Research Article

Delayed Spiking Neural *P* Systems with Scheduled Rules

Qianqian Ren and Xiyu Liu

Academy of Management Science, Business School, Shandong Normal University, Jinan, China

Correspondence should be addressed to Xiyu Liu; xyliu@sdnu.edu.cn

Received 27 May 2020; Revised 15 December 2020; Accepted 24 March 2021; Published 10 April 2021

Academic Editor: Xinzhi Liu

Due to the inevitable delay phenomenon in the process of signal conversion and transmission, time delay is bound to occur between neurons. Therefore, it is necessary to introduce the concept of time delay into the membrane computing models. Spiking neural *P* systems (SN *P* systems), as an attractive type of neural-like *P* systems in membrane computing, are widely followed. Inspired by the phenomenon of time delay, in our work, a new variant of spiking neural *P* systems called delayed spiking neural *P* systems (DSN *P* systems) is proposed. Compared with normal spiking neural *P* systems, the proposed systems achieve time control by setting the schedule on spiking rules and forgetting rules, and the schedule is also used to realize the system delay. A schedule indicates the time difference between receiving and outputting spikes, and it also makes the system work in a certain time, which means that a rule can only be used within a specified time range. We specify that each rule is performed only in the continuous schedule, during which the neuron is locked and cannot send or receive spikes. If the neuron is not available at a given time, it will not receive or send spikes due to the lack of a schedule for this period of time. Moreover, the universality of DSN *P* systems in both generating and accepting modes is proved. And a universal DSN *P* system having 81 neurons for computing functions is also proved.

1. Introduction

As an important branch of natural computing, membrane computing enriches the theoretical framework of biomolecular computing. It is inspired by the biological structures and functions of cells and tissues and has good distributed and parallel computing capabilities. At present, there are four main types of membrane systems, including cell-like *P* systems [1, 2], tissue *P* systems [3, 4], numerical *P* systems [5, 6], and neural *P* systems [7]. Among them, spiking neural *P* systems that belong to neural *P* systems, known as the third-generation neural network model in membrane computing, are inspired by the biological phenomenon that neurons communicate with each other through spikes. In recent years, SN *P* systems have received great attention, and many variants of SN *P* systems have been discussed.

SN *P* systems were first proposed in 2006 [8]. An SN *P* system can be regarded as a directed graph in which nodes represent neurons and arcs represent synapses. It is considered that each neuron mainly contains two components: spikes and rules, where rules include spiking rules $(E/a^{\alpha}) \longrightarrow a; d$ and forgetting rules $a^{s} \longrightarrow \lambda$. They have the different operation. Assuming that the spiking rule is used at time t, it will consume α $(\alpha \geq 1)$ spikes and produce one spike in the neuron at time $t + d$, and then the produced spike will be transmitted to the adjacent neurons through the synapses. The function of the forgetting rule is to eliminate s spikes. Over the years, many variations of the SN *P* systems have been proposed considering different biological characteristics and phenomena, such as SN *P* systems with astrocytes [9], SN *P* systems with neuron division and dissolution [10], cell-like SN *P* systems [11], numerical SN *P* systems [12], and SN *P* systems with polarizations [13].

Because of SN *P* systems with strong expansibility, in recent years, the research on SN *P* system mainly focuses on the theoretical aspect, especially the establishment of various computing models. Many new variants of SN *P* systems are proposed by changing rules, synapses, and structures. Some studies have changed in forms of rules, for instance, SN *P* systems with white hole neurons [14], SN *P* systems with

request rules [15], SN P systems with inhibitory rules [16], SN P systems with communication on request [17, 18], nonlinear SN P systems [19], and SN P systems with target indications [20]. According to the inhibitory way of communication between neurons, many researches about the SN P systems with antispikes are proposed [21, 22]. There are some changes in synapses between neurons, for example, SN P systems with multiple channels [23], SN P systems with inhibitory synapses [24], SN P systems with rules on synapses [25, 26], and SN P systems with thresholds [27], as well as SN P systems with scheduled synapses [28, 29]. According to the self-organizing and adaptive characteristics of artificial neural network, SN P systems with plasticity structure were established [30, 31]. Abstracted from neural network models, some new neural P systems were proposed [32, 33]. Moreover, motivated by the characteristics of dendrites of nerve cells, dendrite P systems were investigated [34].

At present, application research on SN P systems is weak. However, in order to solve practical problems, many scholars have done researches on the application of membrane computing and obtained some research results in some aspects, for example, optimization problems [35, 36], fault diagnosis [37, 38], and image recognition [39, 40]. Most of the research on the application is to combine the optimization algorithm with the P system model and design an approximate optimization algorithm by using the P system, which is mainly used to solve the clustering problem [41, 42]. At present, there are some studies on the combination of P system and neural network, and some achievements have been made in the realization of image processing. The development of the application of membrane computing has been paid much attention. It is a new expansion and breakthrough in the field of membrane computing and develops the research on theory and application.

In the theoretical study of the P systems, their computing power and efficiency are analysed and discussed. The main purpose of computing efficiency is to analyse whether the proposed system can solve the difficult computing problems, such as the traveling salesman problem [43] and the 0-1 knapsack problem [44], within a feasible time. To prove the computing power, the P systems are compared with the Turing machine to analyse its universality [45, 46]. The theoretical research of spiking neural P systems is mainly focused on the research of computing power which can work in generating and accepting modes, and the proposed SN P system can also simulate the register machine.

Five tuples exist in a register machine M that has the form of $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the initial label, l_h is the final label, and I is the set of three types of instructions with the following forms:

(i) l_i: $(\mathrm{ADD}(r), l_j, l_k)$ is the ADD instruction. When the value stored in the register r is increased 1, it will be moved to one of the instructions with labels l_j and l_k nondeterministically.

(ii) l_i: $(\mathrm{SUB}(r), l_j, l_k)$ is the SUB instruction. The register r is not empty and subtracts 1 to move the instruction with label l_j. If the register r is empty, the value 1 subtracted from register r will be moved to the instruction with label l_k.

(iii) l_h: HALT is the halt instruction. When the process of computation reaches the halt instruction, the computation will stop.

Generally, delays in current SN P systems indicate that the neuron will be closed before a certain time and then reopen after that to accept spikes. But in this paper, the schedule is added on the spiking and forgetting rules to explain the delay, and it is more reasonable to achieve the fact that there is a delay between the input and output. Therefore, delayed SN P systems (DSN P systems) are proposed. The rules are only allowed to run for a specific period of time, which represents the time delay between input and output of spikes. The activation of a reference neuron, that is, the application of a rule, indicates when a rule in a connected neuron takes effect or becomes available. For example, the schedule $[d, d+1)$ means that the rule in the neuron is only available beginning at time d and before time $d+1$. In other words, this rule is activated after time d, producing and sending spikes before time $d+1$.

This work mainly proposes DSN P systems that are a new variant of SN P systems and presents the fact that there is a time difference between input and output in the form of a time interval, called schedule. The schedule is set on spiking rules and forgetting rules to determine the performance of spikes in the neuron. The performance of the system is continuous in total time, and once the rule satisfies the activation condition of the neuron, the neuron will be locked by this rule; that is to say, the rule must be used and can only be used in the specified period of time. It is notable that the classic SN P systems that apply rules are based on the principle of maximal parallelism, but DSN P systems have changed it so that the rules are executed in order according to the schedules. In addition, the computing power of DSN P systems under the mode of generating and accepting is proved. In this way, DSN P systems are Turing universal as the number generating devices and accepting devices.

The structure of the rest of this paper is as follows: Section 2 proposes the definition of DSN P systems and gives an example to illustrate the operation mode of the DSN P system. And Section 3 proves the Turing universality of DSN P systems by proving the computing power of the system in the generating mode and the accepting mode, respectively. Then, a small universal DSN P system for computing functions is proved in Section 4. Finally, Section 5 summarizes the work we have done and explains the following research directions.

2. Delayed Spiking Neural P Systems

In this section, a new variant of spiking neural P system is proposed, called delayed spiking neural P systems (DSN P systems). Firstly, the definition of the system is given, and detailed explanations of the system are also given. And then, an example is given to show how the system works.

2.1. Definition

Definition 1. A DSN P system of degree $m \geq 1$ is defined as

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, \text{ref}, \text{syn}, \text{in}, \text{out}), \quad (1)$$

where

(1) $O = a$ is the alphabet and a indicated the spike.

(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons with the form (n_i, R_i), $n_i \in N^+$, and R_i is the set of rules as the following two forms:

(i) Spiking rules are all in neurons with the form $(E/a^\alpha) \longrightarrow a; [t_1, \tau, t_2]$, where E is a regular expression, $\alpha \geq 1$, and $0 \leq t_1 < \tau \leq t_2$.

(ii) Forgetting rules are all in neurons with the form $(E/a^\beta) \longrightarrow \lambda; [t_1, \tau, t_2]$, where E is a regular expression, $\beta \geq 1$, and $0 \leq t_1 < \tau \leq t_2$.

(3) ref$\subseteq\{1, 2, \ldots, m\}$ is the set of reference neurons.

(4) syn $\in \{1, \ldots, h\} \times \{1, \ldots, h\}$ represents the synapses, for any $(i, j) \in$ syn, $1 \leq i, j \leq h$, and $i \neq j$.

(5) in and out are the input neuron and output neuron, respectively.

How the rules work is explained here. Spiking rules $(E/a^\alpha) \longrightarrow a; [t_1, \tau, t_2]$ can be executed on the condition that the neuron contains at least α spikes and the running time must conform to the schedule. E is a regular expression over a. Each time the rule is applied, α spikes are consumed to produce one spike, which is transmitted to the connected neurons. Assume that σ_i contains b spikes and a a spiking rule. The rule can be applied if and only if $b \geq \alpha$ and the performing time conforms to the schedule of this rule. When the number of spikes in the neuron is less than α, the rule will not be used.

The schedule $[t_1, \tau, t_2)$ represents the neuron output spikes from association neurons before time t_2. t_1 is the time when the neuron receives spikes, and τ indicates the performing time of the rule. If $t_1 \neq \tau$, the neuron will receive spikes at time t_1, and the specific rule in neuron will be performed at time τ. If $t_1 = \tau$, the spiking rules can be used shorthand $(E/a^\alpha) \longrightarrow a; [t_1, t_2]$ and it means the neuron receives spikes while the rule is executed, no longer stagnant. In other words, there is no delay during receiving spikes and the executing of the rule. The schedule makes the rules run in a certain order. And if there is no delay between receiving and transmitting spikes, when spikes are transmitted out of the neuron during the synchronization of the rule execution, spiking rules are written as $(E/a^\alpha) \longrightarrow a$ and forgetting rules can be written as $a^\beta \longrightarrow \lambda$.

Note that the performing time of this system is continuous. Continuity refers to the continuous schedule of each time of the system, and the rule can only be used if it is continuous with the schedule of the superior connected neuron. For instance, the system has three neurons with three rules, each rule has a schedule; that is, $[t_1, t_2) \cup [t_2, t_3] \cup [t_3, t_4] = [t_1, t_4]$. Therefore, we consider that the system is continuous in $[t_1, t_4)$, and if the neuron fires in $[t_1, t_2)$; then, its connected neurons fire only when their rules have the schedule $[t_2, t_3)$. In our figures, we indicate a reference neuron with the symbol "•", denoted as $\sigma_i^{\cdot}$. The reference neuron $\sigma_i^{\cdot}$ is activated at time t_1 and outputs the spike before time t_2 which indicates its rule with schedule $[t_1, t_2)$, the connected neuron is only available beginning at time t_2. If the neuron $\sigma_i^{\cdot}$ applies a rule and produces a spike but no rule in the adjacent neuron is available at the time of outputting, then no neuron can receive this spike.

Forgetting rules $a^\beta \longrightarrow \lambda; [t_1, \tau, t_2)$ represent that the rules can be used only if when there are β spikes in neuron σ_i and the rules also be allowed to use at time t_1. The function of forgetting rules is to consume β spikes in neurons. The schedule for forgetting rules has the same meaning as spiking rules. Each neuron may contain more than one forgetting rule or none. The neuron contains at least one spiking rule or forgetting rule, and spiking rules and forgetting rules can both exist in the neuron. And only one rule can be used in each neuron at each time. All neurons in the spiking neural P system work in parallel, but the rules in each neuron are applied sequentially. In other words, if there are at least two rules in the neuron, then according to the time control over the rule, only one rule can be selected which meets the schedule.

The configuration of system Π at a given moment is composed of the number of spikes contained in each neuron. At the initial moment, every neuron in system Π is in an open state without any rules, denoted as the initial configuration C_0. When each neuron in system Π has no rules which can be used and keep open, the system reaches the halt configuration. System Π evolves from one configuration C_1 to another configuration C_2 by executing the rules in each neuron. Such a process is called the transfer from C_1 to C_2, denoted as $C_1 \Rightarrow C_2$. A calculation of system Π consists of a series of transitions from the initial configuration and the computation will stop if and only if it can reach the halt configuration.

In the generating mode, the spiking neural P system contains at least one output neuron of which the function is to send the generated spikes to the environment. There are several definitions of the calculation result of spiking neural P systems [47, 48]. In this paper, the calculation result is the difference between the first two times that neuron σ_{out} fires and sends spikes to the environment. For instance, t_1 is the time of firing the neuron σ_{out} for the first time, and t_2 is the time of firing the neuron σ_{out} for the second time, so the calculation result is $t_2 - t_1$. Because the system in generating mode is deterministic, it produces a collection of numbers, denoted as $N_2(\Pi)$, where the subscript 2 represents the schedule of the first two spikes. When the spiking neural P system is in the accepting mode, there are input neurons in the system, whose function is to receive spikes from the environment. When the number n is identified by this system, it is encoded into a spike train such as $100 \cdots 01\,(10^n1)$ (the spike train is a binary language defined on $\{0, 1\}$, where 1 represents one spike and 0 represents no spike). And then, the system reads the spike train through the input neurons. The set of all numbers accepted by the system is denoted as $N_{\text{acc}}(\Pi)$.

The set of all the numbers generated or accepted by the DSNP system in generating and accepting modes is denoted as $N_\alpha(\Pi)$, where $\alpha \in \{2, \text{acc}\}$. The family of all sets $N_\alpha(\Pi)$ denotes $N_\alpha\text{DSNP}_n^m$, and it means the DSN P system having no more than m rules and no more than n neurons under the generating or accepting mode. If the values of m and n cannot be calculated, it is denoted by *.

2.2. Illustrative Example. An example which is a DSN P system with five neurons is given in Figure 1. And the number of spikes in the neuron at each time is given in Table 1. In this illustrative example, five neurons are used to explain the way of performance by using spiking rules and forgetting rules. Assume that neuron σ_{g_1} has one spike at time t; it will produce one a and send it to neurons σ_{g_2} and σ_{g_3}, respectively. The neuron σ_{g_3} has the form of rule as $a \longrightarrow a; [1, 2, 3)$, and it means that this rule allows neuron σ_{g_3} to receive one spike that came from neuron σ_{g_1} at time $t + 1$. In this way, at time $t + 1$, σ_{g_2} and σ_{g_3} both have one spike.

Before time $t + 2$, σ_{g_2} sends a spike to neurons σ_{g_3} and σ_{g_4}, respectively. And then neurons σ_{g_3} and σ_{g_4} each receive one spike at time $t + 2$. At the same time, the neuron σ_{g_3} performs the rule $a \longrightarrow a; [1, 2, 3)$ and σ_{g_3} receives one spike from σ_{g_2} which means that the rule $a \longrightarrow \lambda; [2, 3)$ will be used. Meanwhile, the rule $a \longrightarrow a; [2, 3)$ in neuron α_{g4} can be applied. Therefore, two spikes in neuron σ_{g_5} can activate the rule $a^2 \longrightarrow a; [3, 4)$ at time $t + 3$, one from neuron σ_{g_3} and another one from neuron σ_{g_4}. And then one spike will be sent to other connected neurons before time $t + 4$.

3. Computing Power as the Number Generator and Acceptor

This section proves the computing power of systems as generating devices and accepting devices and shows that DSN P systems are Turing universal. In order to prove the computing power of a DSN P system, a register machine $M = (m, H, l_0, l_h, I)$ is considered. Register r in M corresponds to neuron σ_r in system Π. All of the instructions also correspond to neurons in Π, such that an ADD instruction l_i corresponds to neuron σ_{l_i}. And if register r stores number n, $2n$ spikes will be stored in neuron σ_r.

There are two working modes in register machine, the generating mode, and the accepting mode. The system Π in generating mode has three modules: module ADD, module SUB, and module OUTPUT. And in accepting mode, the system has three types of modules: module ADD, module SUB, and module INPUT. The set of numbers generated by the register machine M is denoted as $N(M)$. In both modes, a Turing computable set of natural numbers can be generated; that is, $N(M) = \text{NRE}$. Here, NRE refers to the set of numbers calculated by the Turing machine.

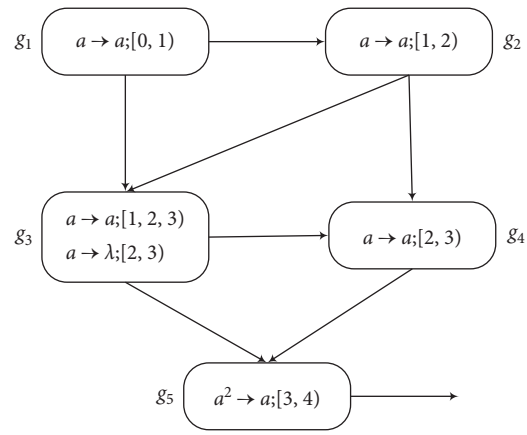

Figure 1: An example of the DSN P system working.

Table 1: The results of the example.

	σ_{g_1}	σ_{g_2}	σ_{g_3}	σ_{g_4}	σ_{g_5}
t	1	0	0	0	0
$t + 1$	0	1	1	0	0
$t + 2$	0	0	$1 + 1$	1	0
$t + 3$	0	0	0	0	2
$t + 4$	0	0	0	0	0

3.1. The DSN P System as Generator

Theorem 1. $N_{gen}DSNP_*^2 = NRE$.

Proof. In generating mode, all registers are empty in the initial state, starting with instruction l_0. The number stored in register 1 at the end of the calculation is the number generated by the register. Assume all neurons are empty, except neuron $\sigma_{l_0}^\bullet$ which has one spike. The module ADD is used to simulate ADD instruction l_i: $(\text{ADD}(r), l_j, l_k)$, shown in Figure 2. The reference neuron in this module is neuron σ_{l_i}, denoted as $\sigma_{l_i}^\bullet$. Assume the rule in neuron $\sigma_{l_i}^\bullet$ is activated at time t, and it will send one spike to neurons σ_{b_1} and σ_{b_2}, respectively. At time $t + 1$, neurons σ_{b_1} and σ_{b_2} receive this spike. Meanwhile, the rule in σ_{b_2} is nondeterministic chosen one to perform, so there are two cases:

(1) At time $t + 1$, if the rule $a \longrightarrow a; [1, 2)$ is used, the neuron σ_{b_1} will accept one spike from neuron σ_{b_1} at time $t + 2$, so that the rule $a \longrightarrow a; [2, 3)$ in neuron σ_{b_1} can be activated. At the same time, neuron σ_{l_k} receives one a from neuron σ_{b_2} at time $t + 2$, and the rule $a \longrightarrow a; [1, 2, 3)$ in neuron σ_{b_1} is applied. At time $t + 3$, both σ_{b_3} and σ_r receive two spikes from neuron σ_{b_1}, so the rule $a^2 \longrightarrow \lambda; [3, 4)$ can be used. Therefore, the neuron σ_{l_j} is empty, and neuron σ_r has two spikes.

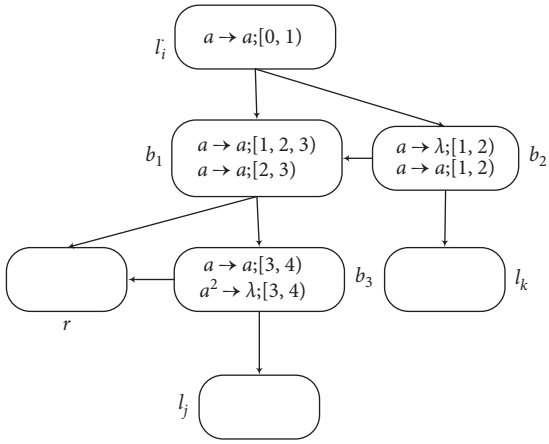

Figure 2: Module ADD: simulating the ADD instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$.

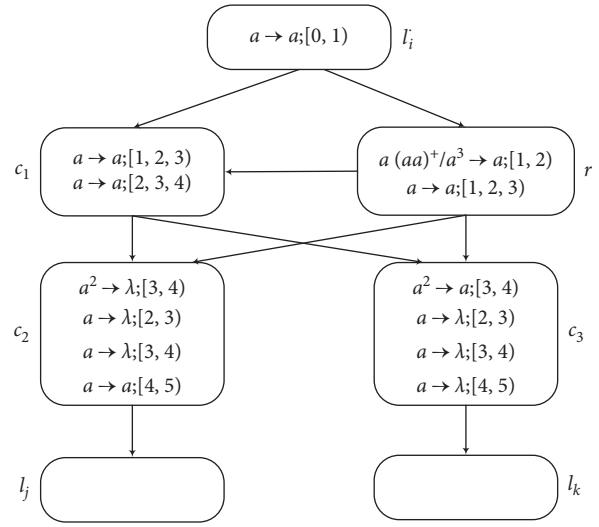

Figure 3: Module SUB: simulating the SUB instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$.

(2) At time $t + 1$, if the rule $a \longrightarrow \lambda$; $[1, 2)$ is applied, then the neuron σ_{l_k} is empty at time $t + 2$, and the rule $a \longrightarrow a$; $[1, 2, 3)$ in neuron σ_{b_1} is enabled. Each of neurons σ_{b_3} and σ_r receives one spike from neuron σ_{b_1} at time $t + 3$. So the rule $a \longrightarrow a$; $[3, 4)$ in neuron σ_{b_3} can perform, and then, at time $t + 4$, neurons σ_{l_j} and σ_r receive one spike, respectively. In this way, there are two spikes in the neuron σ_r corresponding to increasing the register r by 1.

So far, the module ADD has been proved to be able to simulate correctly. The following is t the simulating proof of module SUB. The structure and rules of module SUB are shown in Figure 3, which is used to simulate SUB instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$. The simulation of module SUB also starts at reference neuron $\sigma_{l_i^\bullet}$. The neuron $\sigma_{l_i^\bullet}$ activates At time t and sends one spike to neurons σ_r and σ_{c_1}, respectively, before time $t + 1$. It means that neurons σ_r and σ_{c_1} receive this spike at time $t + 1$. Depending on whether the neuron σ_r has spikes at the initial time, there are two cases as follows:

(1) If the neuron σ_r is not empty, it has $2n$ spikes. So, neuron σ_r has $2n + 1$ spikes (odd number) at time $t + 1$, and the rule $(a\,(aa)^+/a^3) \longrightarrow a$; $[1, 2)$ can be used. At time $t + 2$, both neurons σ_{c_3} and σ_{c_2} receive one spike from the neuron σ_r, and the rules $a \longrightarrow \lambda$; $[2, 3)$ in neurons σ_{c_2} and σ_{c_3} can be used. At the same time, σ_{c_1} also receives one spike from σ_r; that is, the rule $a \longrightarrow a$; $[2, 3, 4)$ in neuron σ_{c_1} can be used. At time $t + 3$, neurons σ_{c_2} and σ_{c_3} get one spike from neuron σ_{c_1}, such that rules $a \longrightarrow \lambda$; $[3, 4)$ in σ_{c_2} and σ_{c_3} are performed. At the next time, each of neurons σ_{c_2} and σ_{c_3} receives one spike from σ_{c_1}, and rule $a \longrightarrow a$; $[4, 5)$ in σ_{c_2} and rule $a \longrightarrow \lambda$; $[4, 5)$ in σ_{c_3} are applied. In this way, neuron σ_{l_k} is empty without receiving any spikes and σ_{l_j} gains one spike.

(2) There is no spike in neuron σ_r. At time $t + 1$, rules $a \longrightarrow a$; $[1, 2, 3)$ allow neurons σ_r and σ_{c_1} to obtain one spike from neuron $\sigma_{l_i^\bullet}$, and this rule is activated at time $t + 2$. And then, each of neurons σ_{c_2} and σ_{c_3} obtains two spikes which come from σ_{c_1} and σ_r, so the rule $a^2 \longrightarrow \lambda$; $[3, 4)$ in σ_{c_2} and the rule $a^2 \longrightarrow a$; $[3, 4)$ in σ_{c_3} can fire. Therefore, σ_{l_k} receives one spike from σ_{c_3}, and neuron σ_{l_j} is empty after time $t + 4$.

In particular, there is no interference between SUB modules, even if there are multiple SUB instructions with a common register. An illustrative example is given in Figure 4. Assume two SUB instructions l_i: $(\mathrm{SUB}(r), l_j, l_k)$, and l_v: $(\mathrm{SUB}(r), l'_j, l'_k)$ are simulated. When neuron σ_r fires, neurons $\sigma_{c'_3}$ and $\sigma_{c'_2}$ get one spike at time $t + 2$ or time $t + 3$. But only rules $a \longrightarrow \lambda$; $[2, 3)$ and $a \longrightarrow \lambda$; $[3, 4)$ in neurons $\sigma_{c'_3}$ and $\sigma_{c'_2}$ are enabled. Neurons $\sigma_{l'_j}$ and $\sigma_{l'_k}$ are empty. Therefore, the SUB instruction can be simulated correctly by module SUB.

The structure of module OUTPUT, shown in Figure 5, is used to halt the computation. At the initial state, neuron σ_{out} has one spike. Assume that the neuron σ_1 has $2n$ spikes according to the number n storing in register 1 of M and neuron $\sigma_{l_h}^\bullet$ receives one spike at time t. So the neuron σ_1 gets one spike at time $t + 1$. Thus, the neuron σ_1 has $2n + 1$ spikes, which is odd number, and at time $t + 2$, the rule $a\,(aa)^+ \longrightarrow a$ is activated and sends a spike to neurons σ_{d_1} and σ_{out}, respectively. In this way, σ_{out} has two spikes, which is even number. The rule in σ_{out} can be applied the next time and sends one spike to environment. At the same time, neuron σ_{d_1} sends one spike to neuron σ_{out} and neuron σ_1 fires again. At present, there are three spikes in neuron σ_{out}. No rules in σ_{out} can be enabled since the number of spikes is odd.

After $n - 1$ times, that is, at time $t + n + 1$, neuron σ_1 has only one spike left, so that the rule in σ_1 can no longer be used. At this moment, neuron σ_{d_1} has one spike. Therefore, at time $t + n + 2$, σ_{d_1} is activated and sends a spike to σ_{out}. The number of spikes in σ_{out} is $2n$ which is an even number, so the rule can be applied and sends a spike to environment a second time. The number computed by the DSN P system is the difference between the first two times which send spikes

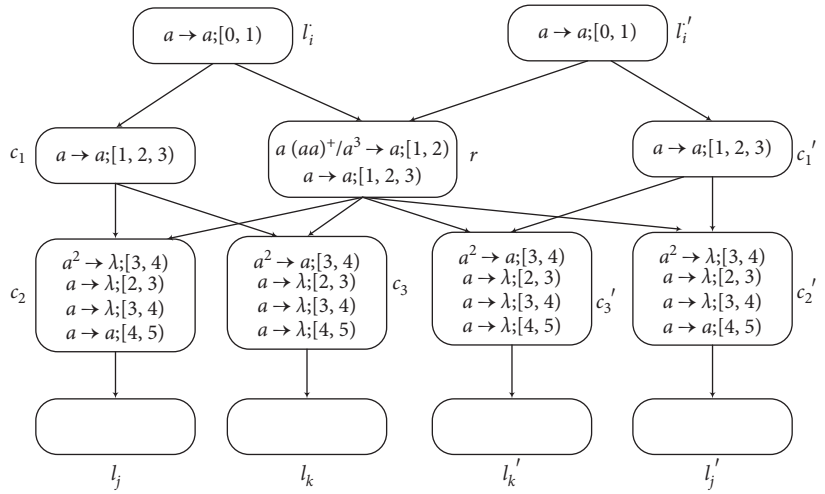

Figure 4: Module SUB-SUB: simulating the SUB instructions l_i: $(\mathrm{SUB}(r), l_j, l_k)$ and l_v: $(\mathrm{SUB}(r), l_j', l_k')$.

to environment; that is, $(t + n + 2) - (t + 2) = n$. The results of each time are shown in Table 2.

These three types of modules simulate the corresponding instructions correctly; that is, system Π correctly simulates the register machine M. Therefore, the conclusion of Theorem 1 is proved. □

3.2. The DSN P System as Acceptor

Theorem 2. $N_{acc}DSNP_*^2 = NRE$.

Proof. In accepting mode, all registers except register 1 are empty in the initial state. As the proof of Theorem 1, the proof of Theorem 2 also needs to simulate register machine M', which is deterministic register machine. Similarly, register r corresponds to neuron σ_r, and if the number n is stored in register r, then there are $2n$ spikes in neuron σ_r. If the register machine reaches the halting instruction l_h, when the calculation stops, the number stored in register 1 is accepted by the register machine. And the proof of module SUB is already proved in Theorem 1. Therefore, in the proof of Theorem 2, we only need to prove whether the modules ADD′ and INPUT can be simulated correctly.

At the initial configuration of Π, all neurons are empty, except for reference neuron $\sigma_{l_0}^{\bullet}$ which contains one spike. Module ADD' shown in Figure 6 differs from model ADD shown in Figure 2. Module ADD′ simulates the ADD′ instruction l_i: $(\mathrm{ADD}(r), l_j)$. Assume that neuron $\sigma_{l_i}^{\bullet}$ has one spike and activates at time t. Thus, each of neurons σ_r and σ_{m_1} receives one spike from σ_{l_i} at time $t + 1$. And the rule in neuron σ_{m_1} can be applied. At time $t + 2$, both neurons σ_r and σ_{l_j} receive a spike, and at this time, neuron σ_r has two spikes. The simulation of module ADD' is complete, since there are two spikes in σ_r and the content of register r increases one.

Module INPUT, shown in Figure 7, is used to compute number n introduced by neuron σ_{in} into σ_1. The module INPUT computes number n between the time of receiving two spikes and the function of this module is to read the

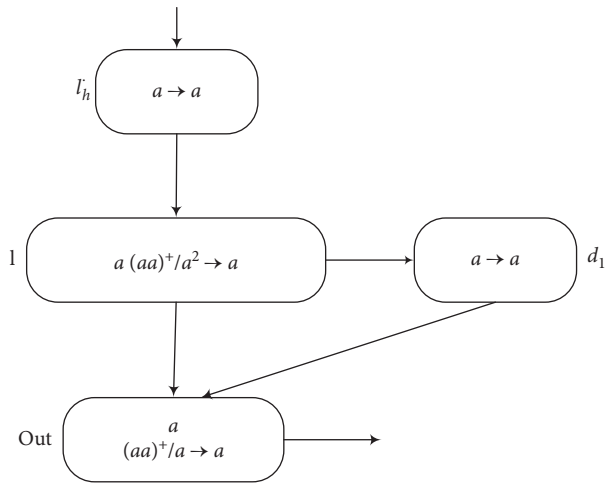

Figure 5: Module OUTPUT.

Table 2: The results of module OUTPUT.

	σ_{l_h}	σ_1	σ_{d_1}	σ_{out}
t	1	$2n$	0	1
$t + 1$	0	$2n + 1$	0	1
$t + 2$	0	$2\ (n - 1) + 1$	1	2 (fire)
$t + 3$	0	$2\ (n - 2) + 1$	1	3
$t + 4$	0	$2\ (n - 3) + 1$	1	5
⋯	⋯	⋯	⋯	⋯
$t + n + 1$	0	1	1	$2n - 1$
$t + n + 2$	0	1	0	$2n$ (fire)

spike train $10^{n-1}1$, where $1 \le n \in N^+$. Assume the neuron σ_{in} receives a spike at time t. Each of neurons σ_{f_2}, σ_{f_3}, and σ_{f_1} receives one spike at time $t + 1$. At the next time, rules in neurons σ_{f_2} and σ_{f_3} are enabled; thus, one spike is received by both neurons σ_{f_3} and σ_1 which comes from the neuron σ_{f_2}. And at the same time, each of neurons σ_{f_2} and σ_1 receives a spike from neuron σ_{f_3}. Therefore, neuron σ_1 has two spikes at that moment. Notably, rules in neurons σ_{f_3} and σ_{f_2} do not stop until σ_{in} received the second spike. After

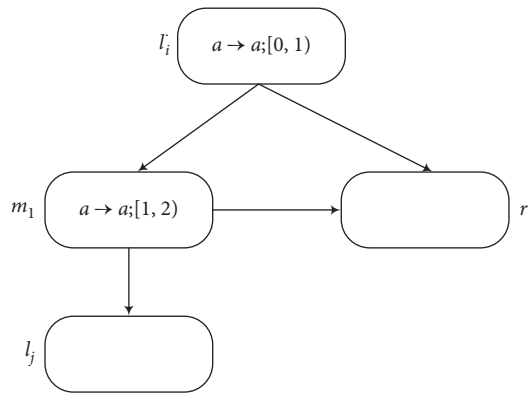

Figure 6: Module ADD': simulating the ADD′ instruction l_i: $(\mathrm{ADD}(r), l_j)$.

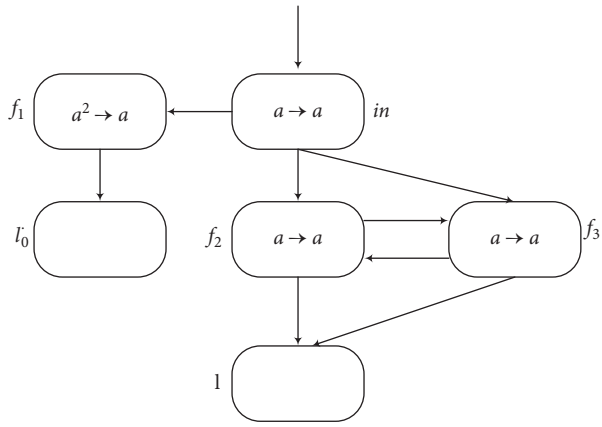

Figure 7: Module INPUT: reading the spike train into the DSN P system.

$n + 1$ times, the neuron σ_1 gets $2n$ spikes, that is, the number n is stored in register 1. And neuron σ_{f_1} has two spikes at this moment. Thus, the rule $a^2 \longrightarrow a$ in σ_{f_1} can be used to produce one spike and send it to neuron $\sigma_{l_0}^{\bullet}$. The results of this model of how to work at each time are presented in Table 3.

From the above proof, module ADD', module SUB, and module INPUT can be correctly simulated. In this way, the proof of Theorem 2 is complete. □

4. A Small Universal DSN *P* System as Computing Functions

This section mainly proves the small universality of the DSN P system for computing functions. First, we need some simple theoretical explanations. Turing computable functions can be computed by register machine M. Here, the way of how a register machine M calculates a function $f: N^K \longrightarrow N$ is described. Register machine M stores K parameters $n_1, n_2, \ldots, n_k$ in the first K registers, and the other registers are empty. We start the calculation with instruction l_0 and end with instruction l_h. When the computation stops, the result is stored in one of the specified

Table 3: The results of module INPUT.

	σ_{in}	σ_{f_2}	σ_{f_3}	σ_1	σ_{f_1}	σ_{l_0}
t	1	0	0	0	0	0
$t+1$	0	1	1	0	1	0
$t+2$	0	1	1	2	1	0
$t+3$	0	1	1	4	1	0
...	...	...	...	...	...	...
$t+n$	1	1	1	$2\ (n-1)$	1	0
$t+n+1$	0	2	2	$2n$	2	0
$t+n+2$	0	2	2	$2n$	0	1

registers, with the rest of the registers empty. It is notable that the register machine M is deterministic when in computing mode. In this paper, we prove the universality of the system as computing functions by simulating a universal register machine M_u.

When proving the small universality of the DSN P system as a computing function device, it is usually considered to use a small universal register machine M_u: $(8, H, l_0, l_h, I)$, more details explained in [49]. M_u contains 8 registers and 23 instructions, as shown in Figure 8. And the key to proving the universality of the system is to find a specific number of neurons that can simulate the register machine M_u and minimize the number through the combination of some ADD and SUB instructions. However, since the SUB instruction l_{19} is related to register 0 which is used to store the calculated results, it is not reasonable to simulate M_u. To solve this problem, a new register 8 is added as the output register, whose function is to save the calculated results. Therefore, register machine M_u: $(8, H, l_0, l_h, I)$ is extended to M_u': $(9, H, l_0, l_h', I)$ and replaces halting instruction l_h with the following three new instructions:

$$\begin{aligned} l_{22}&: \ (\mathrm{SUB}(0), l_{23}, l_h'), \\ l_{23}&: \ (\mathrm{ADD}(8), l_{22}) \\ l_h'&: \ \mathrm{HALT}. \end{aligned} \tag{2}$$

In this way, register machine M_u' contains 9 registers $(0, \ldots, 8)$ and 25 instructions (14 SUB instructions, 10 ADD instructions, and 1 halting instruction).

Theorem 3. *There is a universal DSN P system with 81 neurons for computing functions.*

The specific working process of the DSN P system is shown in Figure 9, including five types of modules: INPUT module, ADD modules, SUB modules, OUTPUT module, and combination modules whose function is to simulate the combination of SUB instructions and ADD instructions. First, the module INPUT reads the encoded parameters of function f into the system. And then, the process enters the register machine simulator. In addition, It is important to note that ADD instructions l_i: $(\mathrm{ADD}(r), l_j)$ are used in module ADD′ as shown in Figure 6 when simulating register machine M_u', and module SUB is shown in Figure 3. The final process is the module OUTPUT, which is structured as shown in Figure 5. But the output register is register 8 instead of register 1.

l_0 : (*SUB* (1), l_1, l_2),
l_1: (*ADD* (7), l_0),
l_2 : (*ADD* (6), l_3),
l_3 : (*SUB* (5), l_2, l_4),
l_4 : (*SUB* (6), l_5, l_3),
l_5 : (*ADD* (5), l_6),
l_6 : (*SUB* (7), l_7, l_8),
l_7 : (*ADD* (1), l_4),
l_8 : (*SUB* (6), l_9, l_0),
l_9 : (*ADD* (6), l_{10}),
l_{10} : (*SUB* (4), l_0, l_{11}),
l_{11} : (*SUB* (5), l_{12}, l_{13}),
l_{12} : (*SUB* (5), l_{14}, l_{15}),
l_{13} : (*SUB* (2), l_{18}, l_{19}),
l_{14} : (*SUB* (5), l_{16}, l_{17}),
l_{15} : (*SUB* (3), l_{18}, l_{20}),
l_{16} : (*ADD* (4), l_{11}),
l_{17} : (*ADD* (2), l_{21}),
l_{18} : (*SUB* (4), l_0, l_h),
l_{19} : (*SUB* (0), l_0, l_{18}),
l_{20} : (*ADD* (0), l_0),
l_{21} : (*ADD* (3), l_{18}),
l_h : *HALT*

Figure 8: A universal register machine M_u: containing 8 registers and 23 instructions.

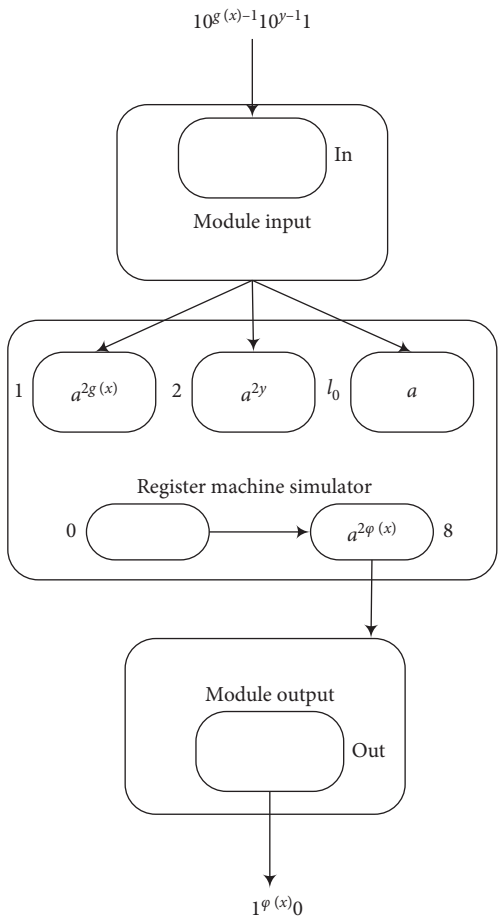

Figure 9: A framework of a DSN P system as computing functions.

In addition, the construction of module INPUT used in the proof of this theorem is different from the previous in accepting devices. Assume that $(\theta_0, \theta_1, \ldots)$ is the fixed parameters of a function, and for any $\theta_x(y)$, register machine M_u' satisfies $\theta_x(y) = M_u'(g(x), y)$. In register machine M_u', register 1 stores parameter $g(x)$, and register 2 stores parameter y. Therefore, the module INPUT is constructed which is shown in Figure 10. In the beginning, the two parameters $g(x)$ and y are encoded into the form of a spike train $10^{g(x)-1}10^{y-1}1$. Then, through the computing of module INPUT, the two parameters $g(x)$ and y are stored in neurons σ_1 and σ_2 replaced by $2g(x)$ spikes and $2y$ spikes, respectively.

Module INPUT shown in Figure 10 is used to compute the function $10^{g(x)-1}10^{y-1}1$ that is introduced by neuron σ_{in} into σ_1 with $2g(x)$ spikes and σ_2 with $2y$ spikes. Assume that the neuron σ_{in} activates at time t. Each of neurons σ_{m_1}, σ_{m_2}, σ_{m_3}, σ_{m_4}, and σ_{m_5} receives one a at time $t+1$. Thus, neurons σ_{m_3} and σ_{m_2} activate, and then neuron σ_{m_2} sends one spike to σ_1 and σ_{m_3}, respectively. Meanwhile, neuron σ_{m_3} sends one spike to σ_1 and σ_{m_2}, so that neuron σ_1 receives two spikes at time $t+2$, and each of neurons σ_{m_3} and σ_{m_2} has one spike. Until the neuron σ_{in} receives the second spike, that is, after $g(x)-1$ times, rules in σ_{m_3} and σ_{m_2} cannot fire. In this way, neuron σ_1 gets $2g(x)$ spikes corresponding to the number $g(x)$ stored in register 1.

Simultaneously, each of neurons σ_{m_1}, σ_{m_4}, and σ_{m_5} has two spikes when neuron σ_1 gets $2g(x)$ spikes. Thus, the rules in σ_{m_4} and σ_{m_5} are enabled, both consuming one spike. And at the next time, neuron σ_2 receives two spikes, one from σ_{m_4} and the other from σ_{m_5}. At the same time, neurons σ_{m_4} and σ_{m_5} each receive a spike from the other, so the number of spikes they contain returns to an even number. In this way, neurons σ_{m_4} and σ_{m_5} remain active for the next $y-1$ times. Therefore, neuron σ_2 accepts $2y$ spikes altogether until neuron σ_{in} receives the third spike. And then, neuron σ_{m_1} has three spike, and the rule $a^3 \longrightarrow a$ can perform and sends one spike to $\sigma_{l_0}^{\bullet}$.

Therefore, the system needs 94 neurons to simulate the universal register machine M_u' including

(i) 6 neurons for module INPUT
(ii) 1×10 neurons for 10 ADD instructions
(iii) 3×14 neurons for 14 SUB instructions
(iv) 25 neurons for 25 labels of instructions
(v) 9 neurons for 9 register
(vi) 2 neurons for module OUTPUT

The module SUB-ADD-1 is simulating the sequence of SUB instructions l_i: $(\text{SUB}(r_1), l_j, l_k)$ and ADD instructions l_j: $(\text{ADD}(r_2), l_g)$, as shown in Figure 11, where $r_1 \neq r_2$. The simulation of module SUB-ADD-1 also starts at reference neuron $\sigma_{l_i^{\bullet}}$. When neuron σ_{l_j} has one spike, ADD instruction is triggered and performed immediately. Thus, the final results should be that neuron σ_{l_g} gains one spike, the register r_2 increases by one, and σ_{l_k} is empty. If neuron σ_{l_k} has one spike, the ADD instruction will not activate. The register r_2 and neuron σ_{l_g} are certainly empty. The computation process is divided into two cases depending on the number of spikes in neuron σ_{r_1}. The process is similar to module SUB (shown in Figure 3). Assume that the reference neuron $\sigma_{l_i^{\bullet}}$ fires at time t.

(1) If the register r_1 is nonempty, neuron σ_{r_1} has $2n$ spikes. At time $t+1$, neuron σ_r has $2n+1$ spikes, and the rule $(a(aa)^+/a^3) \longrightarrow a; [1,2)$ fires. At the same time, neuron σ_{c_1} also gains one spike. Thus, at the next time, each of neurons σ_r, σ_{c_3}, and σ_{c_2} gains one spike, and the rule $a \longrightarrow \lambda; [2,3)$ in neuron σ_{c_2} and $a \longrightarrow \lambda; [2,3)$

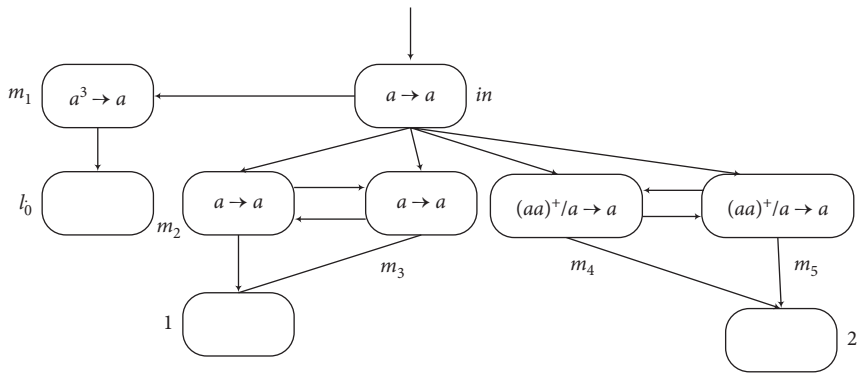

Figure 10: Module INPUT.

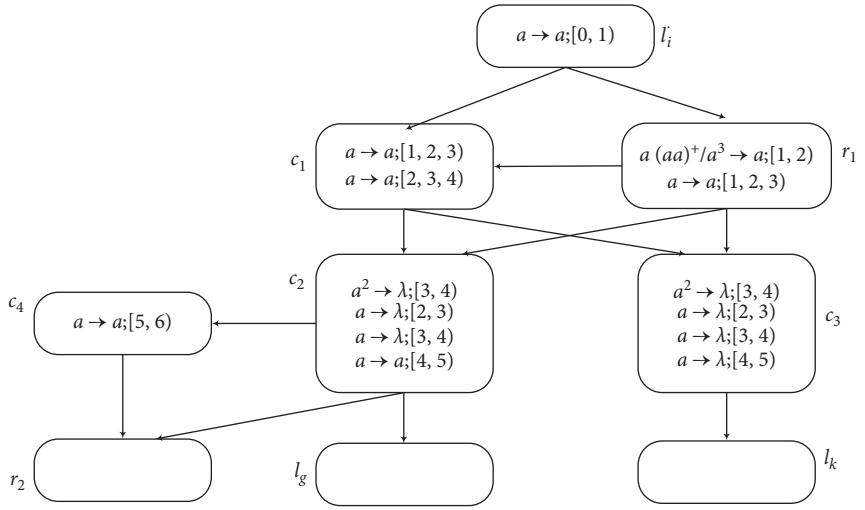

Figure 11: Module SUB-ADD-1: simulating the consecutive SUB instruction l_i: $(\mathrm{SUB}(r_1), l_j, l_k)$ and ADD instruction l_j: $(\mathrm{ADD}(r_2), l_g)$.

in σ_{c_3} can be used. At time $t + 3$, neurons σ_{c_3} and σ_{c_2} gain one spike form σ_{c_1}, and rule $a \longrightarrow \lambda$; $[3, 4)$ can be applied. And at the next time, neurons σ_{c_3} and σ_{c_2} get one spike form σ_{c_1} at again, thus, the rule $a \longrightarrow a$; $[4, 5)$ in σ_{c_2} and $a \longrightarrow \lambda$; $[4, 5)$ in σ_{c_3} fire. At time $t + 5$, σ_{c_4} applies its rule $a \longrightarrow a$; $[5, 6)$ and sends one spike to σ_{r_2} before time $t + 6$. In this way, σ_{r_2} has two spikes, and neuron σ_{l_k} is empty.

(2) If the register r_1 is empty, it means that neuron σ_{r_1} is empty at the initial time. At time $t + 1$, neurons σ_r and σ_{c_1} have one spike, and the rule $a \longrightarrow a$; $[1, 2, 3)$ in σ_r can be applied which means neuron σ_r accepts the spike at this moment. At time $t + 2$, the rule $a \longrightarrow a$; $[1, 2, 3)$ in σ_{c_1} fires and sends one spike to σ_{c_3} and σ_{c_2} before $t + 3$, respectively. At time $t + 3$, σ_{c_3} and σ_{c_2} receive two spikes, and neuron σ_{c_2} chooses to perform $a^2 \longrightarrow \lambda$; $[3, 4)$ and neuron σ_{c_3} performs rule $a^2 \longrightarrow a$; $[3, 4)$. Therefore, σ_{l_k} gains the spike at time $t + 4$, and σ_{l_g} and σ_{r_2} are empty.

In this way, consecutive SUB-ADD instructions can be simulated correctly, and neuron σ_{l_j} can be saved. Then, there are 6 pairs of instructions corresponding to module SUB-ADD-1, which can save 6 neurons associated with labels l_1, l_5, l_7, l_9, l_{16}, and l_{22}.

$$
\begin{aligned}
l_0&: (\mathrm{SUB}(1), l_1, l_2),\\
l_1&: (\mathrm{ADD}(7), l_0),\\
l_4&: (\mathrm{SUB}(6), l_5, l_3),\\
l_5&: (\mathrm{ADD}(5), l_6),\\
l_6&: (\mathrm{SUB}(7), l_7, l_8),\\
l_7&: (\mathrm{ADD}(1), l_4),\\
l_8&: (\mathrm{SUB}(6), l_9, l_0),\\
l_9&: (\mathrm{ADD}(6), l_{10}),\\
l_{14}&: (\mathrm{SUB}(5), l_{16}, l_{17}),\\
l_{16}&: (\mathrm{ADD}(4), l_{11}),\\
l_{22}&: (\mathrm{SUB}(1), l_{23}, l_h'),\\
l_{23}&: (\mathrm{ADD}(8), l_{22}).
\end{aligned}
\tag{3}
$$

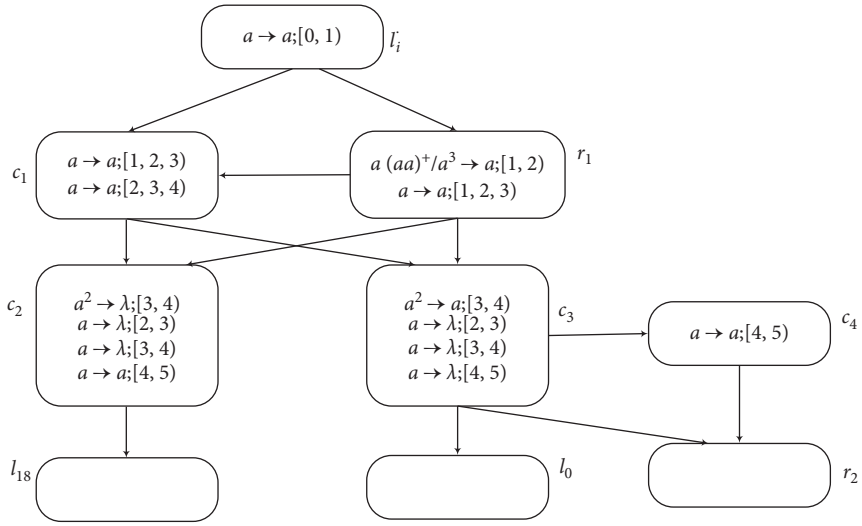

Figure 12: Module SUB-ADD-2: simulating the consecutive SUB instruction l_{15}: $(\mathrm{SUB}(3), l_{18}, l_{20})$ and ADD instruction l_{20}: $(\mathrm{ADD}(0), l_0)$.

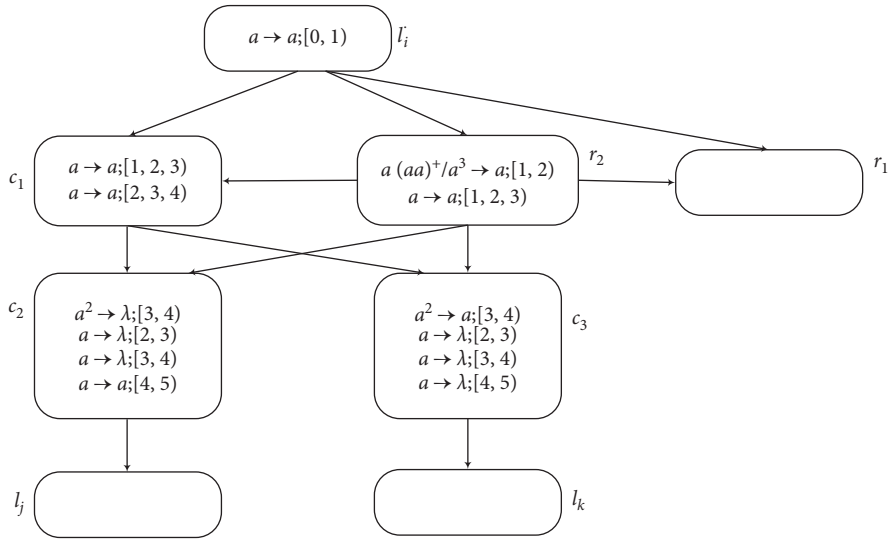

Figure 13: Module ADD-SUB: simulating the sequence of ADD instruction l_i: $(\mathrm{ADD}(r_1), l_g)$ and SUB instruction l_g: $(\mathrm{SUB}(r_2), l_j, l_k)$.

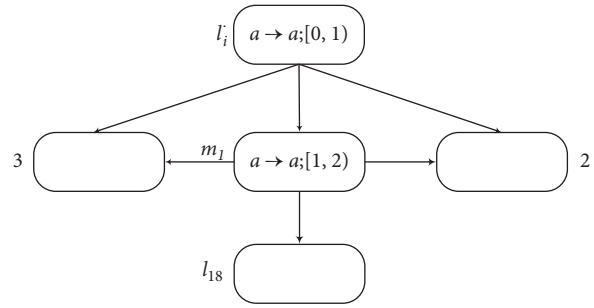

Figure 14: Module ADD-ADD: simulating ADD-ADD instructions l_{17}: $(\mathrm{ADD}(2), l_{21})$ and l_{21}: $(\mathrm{ADD}(3), l_{18})$.

Module SUB-ADD-2 is used to simulate the combination of SUB instruction l_{15}: $(\mathrm{SUB}(3), l_{18}, l_{20})$ and ADD instruction l_{20}: $(\mathrm{ADD}(0), l_0)$. As shown in Figure 12, the computational process of SUB-ADD-2 is similar to that of module SUB-ADD-1. Thus, one neuron associated with l_{20} can be saved.

Module ADD-SUB is shown in Figure 13. And the function of this module is to simulate consecutive ADD-SUB

instructions l_i: $(\mathrm{ADD}(r_1), l_g)$ and l_g: $(\mathrm{SUB}(r_2), l_j, l_k)$, where $r_1 \neq r_2$. Assume that reference neuron $\sigma_{l_i}^{\bullet}$ has one spike at time t, and the produced spike is received by σ_{r_1} and σ_{r_2} at the next time. Then, the working process of next time is the same as module SUB, divided into two cases. σ_{r_1} receives one spike from σ_{r_2} at time $t+2$ or time $t+3$, so that σ_{r_1} has two spikes. Thus, both the ADD and SUB instructions can be implemented correctly by this module. So, neuron σ_{l_g} can be saved, and neuron σ_{m_1} also is saved as compared to module ADD' shown in Figure 6. There are two pairs of instructions that adapt to the module ADD-SUB. Thus, four neurons are saved.

$$\begin{aligned} l_5&: \ (\mathrm{ADD}(5), l_6),\\ l_6&: \ (\mathrm{SUB}(7), l_7, l_8),\\ l_9&: \ (\mathrm{ADD}(6), l_{10}),\\ l_{10}&: \ (\mathrm{SUB}(4), l_0, l_{11}). \end{aligned} \tag{4}$$

ADD-ADD instructions l_{17}: $(\mathrm{ADD}(2), l_{21})$ and l_{21}: $(\mathrm{ADD}(3), l_{18})$ can also be combined by module ADD-ADD (shown in Figure 14). Assume that, at time t, neuron $\sigma_{l_{17}}$ activates. Each of neurons σ_{m_1}, σ_3, and σ_2 receives one spike at time $t+1$. Then, neurons σ_3 and σ_2 get their second spike from σ_{m_1}, at time $t+2$. Thus, the consecutive ADD-ADD instructions can be simulated correctly by module ADD-ADD. Two module ADD can share the common neuron σ_{m_1}, and the neuron associated with label l_{21} is also saved. Therefore, two neurons can be saved by module ADD-ADD.

From the above proof, six neurons are saved by module SUB-ADD-1. One neuron is saved by module SUB-ADD-2. Four neurons are saved by module ADD-SUB. Two neurons are saved by module ADD-ADD. A total of 13 neurons were saved. Therefore, the system can only use 81 neurons to simulate the register machine M_u'.

5. Conclusion

In this paper, a new variant of SN *P* systems, called DSN *P* systems, is proposed, which are inspired by the biological reality that there is a certain time delay between the input and output of neurons. Compared with the classic SN *P* systems, we made corresponding improvements in the spiking rules and forgetting rules by adding schedules to limit the performing time of rules. It achieves a time delay between input and output by scheduled rules and makes the performance of the system continuous in total working time. The rules can only be used within a specified time range, and during this time, the neuron is locked and cannot send or receive spikes. This also allows a neuron with multiple rules to selectively apply the rules according to the schedule when it is activated. In addition, DSN *P* systems have changed the principle (maximal parallelism) of applying rules, and rules are executed in order according to the schedules. And in Theorems 1 and 2, the computational power of DSN *P* systems are proved by simulating the register machine under the generating mode and the accepting mode. In Theorem 3, a universal DSN *P* system having 81 neurons for computing functions is also proved.

In this paper, we only involve the theoretical proof of DSN *P* systems. In the future work, in both theory and application, there are still many problems that can be improved based on DSN *P* systems. In theory, we can further improve DSN *P* systems. We can demonstrate the computational power of systems that work on other modes, such as sequential mode and parallel mode. We can also investigate whether the computational power remains if the types of rules are reduced. And DSN *P* systems with other features are to be studied, such as multiple channels and polarizations. In terms of application, on the one hand, we can consider whether the DSN *P* system can be combined with a neural network and applied to image or text processing, as the work in [50]. On the other hand, DSN *P* systems can also be combined with intelligent algorithm, and the performance of such intelligent optimization systems may be greatly improved. In addition, there are many other directions of application, such as diagnose fault and hardware. Although the application of SN P systems has not been fully studied, as a third-generation neural network, the development of SN *P* systems has infinite possibilities, which need our continuous exploration.

Data Availability

This manuscript does not use any datasets.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (nos. 61876101, 61802234, and 61806114), the Social Science Fund Project of Shandong (16BGLJ06 and 11CGLJ22), China Postdoctoral Science Foundation Funded Project (2017M612339 and 2018M642695), Natural Science Foundation of the Shandong Provincial (ZR2019QF007), China Postdoctoral Special Funding Project (2019T120607), and Youth Fund for Humanities and Social Sciences, Ministry of Education (19YJCZH244).

References

[1] A. A. Mahmood, A. Maroosi, and R. C. Muniyandi, "Membrane computing to enhance time efficiency of minimum dominating set," *Mathematics in Computer Science*, vol. 10, no. 2, pp. 249–261, 2016.

[2] B. Song, M. J. Pérez-Jiménez, and L. Pan, "An efficient time-free solution to sat problem by p systems with proteins on membranes," *Journal of Computer and System Sciences*, vol. 82, no. 2, pp. 1090–1099, 2016.

[3] J. Ming, J. Wang, K. Chen, H. Peng, and X. Song, "Tissue p system combined with producer/consumer and its application in micro-grid economic operation," *Journal of Computational and Theoretical Nanoscience*, vol. 13, no. 6, pp. 3936–3941, 2016.

[4] B. Song, M. J. Pérez-Jiménez, G. Paun, and L. Pan, "Tissue p systems with channel states working in the flat maximally parallel way," *IEEE Transactions on Nanobioscience*, vol. 15, no. 7, pp. 645–656, 2016.

[5] L. Pan, Z. Zhang, T. Wu, and J. Xu, "Numerical p systems with production thresholds," *Theoretical Computer Science*, vol. 673, pp. 30–41, 2017.

[6] L. Liu, W. Yi, Q. Yang, H. Peng, and J. Wang, "Numerical p systems with boolean condition," *Theoretical Computer Science*, vol. 785, pp. 140–149, 2019.

[7] L. Xu and P. Jeavons, "Simple neural-like p systems for maximal independent set selection," *Neural Computation*, vol. 25, no. 6, pp. 1642–1659, 2013.

[8] M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural p systems," *Fundamenta Informaticae*, vol. 71, no. 2–3, pp. 279–308, 2006.

[9] L. Pan, J. Wang, and H. J. Hoogeboom, "Spiking neural p systems with astrocytes," *Neural Computation*, vol. 24, no. 3, pp. 805–825, 2014.

[10] L. Pan, G. Păun, and M. J. Pérez-Jiménez, "Spiking neural p systems with neuron division and budding," *Science China Information Sciences*, vol. 54, no. 8, p. 1596, 2011.

[11] T. Wu, Z. Zhang, G. Păun, and L. Pan, "Cell-like spiking neural p systems," *Theoretical Computer Science*, vol. 623, pp. 180–189, 2016.

[12] T. Wu, L. Pan, Q. Yu et al., "Numerical spiking neural P systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2020, In press.

[13] T. Wu, A. Păun, Z. Zhang, and L. Pan, "Spiking neural p systems with polarizations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3349–3360, 2017.

[14] T. Song, F. Gong, X. Liu, Y. Zhao, and X. Zhang, "Spiking neural p systems with white hole neurons," *IEEE Transactions on Nanobioscience*, vol. 15, no. 7, pp. 666–673, 2016.

[15] T. Song and L. Pan, "Spiking neural p systems with request rules," *Neurocomputing*, vol. 193, pp. 193–200, 2016.

[16] H. Peng, B. Li, J. Wang et al., "Spiking neural p systems with inhibitory rules," *Knowledge-Based Systems*, vol. 188, Article ID 105064, 2020.

[17] L. Pan, G. Păun, G. Zhang, and F. Neri, "Spiking neural p systems with communication on request," *International Journal of Neural Systems*, vol. 27, no. 8, Article ID 1750042, 2017.

[18] T. Pan, X. Shi, Z. Zhang, and F. Xu, "A small universal spiking neural p system with communication on request," *Neurocomputing*, vol. 275, pp. 1622–1628, 2018.

[19] H. Peng, "Nonlinear spiking neural P systems," *International Journal of Neural Systems*, vol. 30, no. 10, Article ID 2050008, 2020.

[20] T. Wu, L. Zhang, and L. Pan, "Spiking neural P systems with target indications," *Theoretical Computer Science*, vol. 862, pp. 250–261, 2020.

[21] X. Song, J. Wang, H. Peng et al., "Spiking neural p systems with multiple channels and anti-spikes," *Biosystems*, vol. 169-170, pp. 13–19, 2018.

[22] Q. Yang, B. Li, Y. Huang, H. Peng, and J. Wang, "Spiking neural p systems with structural plasticity and anti-spikes," *Theoretical Computer Science*, vol. 801, pp. 143–156, 2020.

[23] H. Peng, J. Yang, J. Wang et al., "Spiking neural p systems with multiple channels," *Neural Networks*, vol. 95, pp. 66–71, 2017.

[24] T. Song and X. Wang, "Homogenous spiking neural p systems with inhibitory synapses," *Neural Processing Letters*, vol. 42, no. 1, pp. 199–214, 2015.

[25] T. Song and L. Pan, "Spiking neural p systems with rules on synapses working in maximum spiking strategy," *IEEE Transactions on Nanobioscience*, vol. 14, no. 4, pp. 465–477, 2015.

[26] Y. Su, T. Wu, F. Xu, and A. Păun, "Spiking neural P systems with rules on synapses working in sum spikes consumption strategy," *Fundamenta Informaticae*, vol. 156, no. 2, pp. 187–208, 2017.

[27] X. Zeng, X. Zhang, T. Song, and L. Pan, "Spiking neural p systems with thresholds," *Neural Computation*, vol. 26, no. 7, pp. 1340–1361, 2014.

[28] F. G. C. Cabarle, H. N. Adorna, M. Jiang, and X. Zeng, "Spiking neural p systems with scheduled synapses," *IEEE Transactions on Nanobioscience*, vol. 16, no. 8, pp. 792–801, 2017.

[29] A. Bibi, F. Xu, H. N. Adorna, and F. G. C. Cabarle, "Sequential spiking neural p systems with local scheduled synapses without delay," *Complexity*, vol. 2019, Article ID 7313414, 12 pages, 2019.

[30] F. G. C. Cabarle, H. N. Adorna, M. J. Pérez-Jiménez, and T. Song, "Spiking neural p systems with structural plasticity," *Neural Computing and Applications*, vol. 26, no. 8, pp. 1905–1917, 2015.

[31] F. G. C. Cabarle, R. T. A. de la Cruz, X. Zhang, M. Jiang, X. Liu, and X. Zeng, "On string languages generated by spiking neural p systems with structural plasticity," *IEEE Transactions on Nanobioscience*, vol. 17, no. 4, pp. 560–566, 2018.

[32] H. Peng, J. Wang, M. J. Pérez-Jiménez, and A. Riscos-Núñez, "Dynamic threshold neural P systems," *Knowledge-Based Systems*, vol. 163, pp. 875–884, 2019.

[33] H. Peng and J. Wang, "Coupled neural p systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 6, pp. 1672–1682, 2019.

[34] H. Peng, T. Bao, X. Luo et al., "Dendrite P systems," *Neural Networks*, vol. 127, pp. 110–120, 2020.

[35] G. Zhang, H. Rong, F. Neri, and M. J. Prez-Jimnez, "An optimization spiking neural p system for approximately solving combinatorial optimization problems," *International Journal of Neural Systems*, vol. 24, no. 5, 2014.

[36] M. Zein, A. Adl, and A. E. Hassanien, "Spiking neural p grey wolf optimization system: novel strategies for solving non-determinism problems," *Expert Systems with Applications*, vol. 121, pp. 204–220, 2019.

[37] H. Peng, J. Wang, J. Ming et al., "Fault diagnosis of power systems using intuitionistic fuzzy spiking neural p systems," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4777–4784, 2018.

[38] J. Wang, H. Peng, W. Yu et al., "Interval-valued fuzzy spiking neural p systems for fault diagnosis of power transmission networks," *Engineering Applications of Artificial Intelligence*, vol. 82, pp. 102–109, 2019.

[39] T. Song, S. Pang, S. Hao, A. Rodríguez-Patón, and P. Zheng, "A parallel image skeletonizing method using spiking neural p systems with weights," *Neural Processing Letters*, vol. 50, no. 2, pp. 1485–1502, 2019.

[40] T. Song, L. Pan, T. Wu, P. Zheng, M. L. D. Wong, and A. Rodriguez-Paton, "Spiking neural P systems with learning functions," *IEEE Transactions on Nanobioscience*, vol. 18, no. 2, pp. 176–190, 2019.

[41] H. Peng, X. Luo, Z. Gao, J. Wang, and Z. Pei, "A novel clustering algorithm inspired by membrane computing," *The Scientific World Journal*, vol. 2015, Article ID 929471, 8 pages, 2015.

[42] H. Peng, J. Wang, P. Shi, A. Riscos-Núñez, and M. J. Pérez-Jiménez, "An automatic clustering algorithm inspired by membrane computing," *Pattern Recognition Letters*, vol. 68, pp. 34–40, 2015.

[43] P. Guo, J. Xiang, J. Xie, and J. Zheng, "A P system for solving all-solutions of TSP," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 9, 2017.

[44] X. Liu, Z. Li, J. Suo, Y. Ju, J. Liu, and X. Zeng, "Solving multidimensional 0-1 knapsack problem with time-free tissue p systems," *Journal of Applied Mathematics*, vol. 2014, Article ID 372768, 6 pages, 2014.

[45] T. Wu, F.-D. Bilbie, A. Paun, L. Pan, and F. Neri, "Simplified and yet Turing universal spiking neural P systems with communication on request," *International Journal of Neural Systems*, vol. 28, no. 8, 2018.

[46] T. Wu and L. Pan, "The computation power of spiking neural P systems with polarizations adopting sequential mode induced by minimum spike number," *Neurocomputing*, vol. 401, pp. 392–404, 2020.

[47] G. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, "Spike trains in spiking neural p systems," *International Journal of Foundations of Computer Science*, vol. 17, no. 4, pp. 975–1002, 2006.

[48] H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, G. Păun, and M. J. Pérez-Jiménez, "Spiking neural p systems with extended rules: universality and languages," *Natural Computing*, vol. 7, no. 2, pp. 147–166, 2008.

[49] I. Korec, "Small universal register machines," *Theoretical Computer Science*, vol. 168, no. 2, pp. 267–301, 1996.

[50] B. Li, H. Peng, J. Wang et al., "Multi-focus image fusion based on dynamic threshold neural P systems and surfacelet transform," *Knowledge-Based Systems*, vol. 196, Article ID 105794, 2020.