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 $(E/a^\alpha) \rightarrow a; d$ and forgetting rules $a^\lambda \rightarrow \lambda$. They have the different operation. Assuming that the spiking rule is used at time $t$, it will consume $a (a \geq 1)$ 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 antispike rules 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 approximation 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 $M = (m, H, l_0, l_f, l)$, where $m$ is the number of registers, $H$ is the set of instruction labels, $l_0$ is the initial label, $l_f$ is the final label, and $l$ is the set of three types of instructions with the following forms:

(i) $l_i$: (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) $l_k$: (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 $l_k$.

(iii) $l_k$: 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

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

where

1. $\mathcal{O} = a$ is the alphabet and $a$ indicated the spike.
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons with the form $(n_i, R_i)$, $n_i \in \mathbb{N}^+$, and $R_i$ is the set of rules as the following two forms:
   
   (i) Spiking rules are all in neurons with the form $E(a^\alpha) \rightarrow a; [t_1, t_2]$, where $E$ is a regular expression, $\alpha \geq 1$, and $0 \leq t_1 < t_2$.
   
   (ii) Forgetting rules are all in neurons with the form $E(a^\beta) \rightarrow \lambda; [t_1, t_2]$, where $E$ is a regular expression, $\beta \geq 1$, and $0 \leq t_1 < t_2$.
3. $\text{ref} \subseteq \{1, 2, \ldots, m\}$ is the set of reference neurons.
4. $\text{syn} \in \{1, \ldots, h\} \times \{1, \ldots, h\}$ represents the synapses, for any $(i, j) \in \text{syn}$, $1 \leq i, j \leq h$, and $i \neq j$.
5. $\text{in}$ and $\text{out}$ are the input neuron and output neuron, respectively.

How the rules work is explained here. Spiking rules $(E(a^\alpha) \rightarrow a; [t_1, t_2])$ can be executed on the condition that the neuron contains at least $\alpha$ spikes and the running time must conform to the schedule. $E$ is a regular expression over $a$. Each time the rule is applied, $a$ spikes are consumed to produce one spike, which is transmitted to the connected neurons. Assume that $\sigma_i$ contains $b$ spikes and $a$ 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 $a$, the rule will not be used.

The schedule $[t_1, t_2]$ represents the neuron output spikes from association neurons before time $t_2$. $t_1$ is the time when the neuron receives spikes, and $t_2$ indicates the performing time of the rule. If $t_1 \neq t_2$, the neuron will receive spikes at time $t_2$, and the specific rule in neuron will be performed at time $t$. If $t_1 = t_2$, the spiking rules can be used shorthand $(E(a^\alpha) \rightarrow a; [t_1, t_2])$ 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 $(E(a^\alpha) \rightarrow a$ and forgetting rules can be written as $a^{\beta} \rightarrow \lambda$.

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, $[t_1, t_2] \cup [t_3, t_4] \cup [t_5, t_6] = [t_1, t_6]$. Therefore, we consider that the system is continuous in $[t_1, t_6]$, and if the neuron fires in $[t_1, t_2]$; then, its connected neurons fire only when their rules have the schedule $[t_1, t_2]$. In our figures, we indicate a reference neuron with the symbol "$\ast$", denoted as $\sigma_i$. The reference neuron $\sigma_i$ 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 $\sigma_i$ 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 $a^{\beta} \rightarrow \lambda; [t_1, t_2]$ represent that the rules can be used only if when there are $\beta$ spikes in neuron $\sigma_i$ and the rules also be allowed to use at time $t_1$. The function of forgetting rules is to consume $\beta$ 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 $\Pi$ at a given moment is composed of the number of spikes contained in each neuron. At the initial moment, every neuron in system $\Pi$ is in an open state without any rules, denoted as the initial configuration $C_0$. When each neuron in system $\Pi$ has no rules which can be used and keep open, the system reaches the halt configuration. System $\Pi$ 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 $\Pi$ 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 $\sigma_{out}$ fires and sends spikes to the environment. For instance, $t_1$ is the time of firing the neuron $\sigma_{out}$ for the first time, and $t_2$ is the time of firing the neuron $\sigma_{out}$ 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 $100 \cdots 01 \ast (10^01)$ (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 $N_{acc}(\Pi)$.
The set of all the numbers generated or accepted by the DSNP system in generating and accepting modes is denoted as $N_a(\Pi)$, where $a \in \{2, \text{acc}\}$. The family of all sets $N_a(\Pi)$ denotes $N_a(\text{DSNP}^m)$, and it means the DSNP 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 DSNP 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 $a_{\beta_1}$ has one spike at time $t$; it will produce one $a$ and send it to neurons $a_{\beta_2}$ and $a_{\beta_3}$, respectively. The neuron $a_{\beta_4}$ has the form of rule as $a \rightarrow a; [1, 2, 3]$, and it means that this rule allows neuron $a_{\beta_4}$ to receive one spike that came from neuron $a_{\beta_4}$, at time $t + 1$. In this way, at time $t + 1$, $a_{\beta_2}$ and $a_{\beta_3}$ both have one spike.

Before time $t + 2$, $a_{\beta_2}$ sends a spike to neurons $a_{\beta_1}$ and $a_{\beta_3}$, respectively. And then neurons $a_{\beta_1}$ and $a_{\beta_3}$ each receive one spike at time $t + 2$. At the same time, the neuron $a_{\beta_4}$ performs the rule $a \rightarrow a; [1, 2, 3]$ and $a_{\beta_4}$ receives one spike from $a_{\beta_4}$, which means that the rule $a \rightarrow \lambda; [2, 3]$ will be used. Meanwhile, the rule $a \rightarrow a; [2, 3]$ in neuron $a_{\beta_4}$ can be applied. Therefore, two spikes in neuron $a_{\beta_4}$ can activate the rule $a_1^2 \rightarrow a; [3, 4]$ at time $t + 3$, one from neuron $a_{\beta_4}$ and another one from neuron $a_{\beta_4}$. 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 DSNP systems are Turing universal. In order to prove the computing power of a DSNP system, a register machine $M = (m, H, I_0, I_1, I)$ is considered. Register $r$ in $M$ corresponds to neuron $a_r$ in system $\Pi$. All of the instructions also correspond to neurons in $\Pi$, such that an ADD instruction $a_i$ corresponds to neuron $a_i$. And if register $r$ stores number $n$, $2n$ spikes will be stored in neuron $a_r$.

There are two working modes in register machine, the generating mode, and the accepting mode. The system $\Pi$ 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.

### Table 1: The results of the example.

<table>
<thead>
<tr>
<th>$a_{\beta_1}$</th>
<th>$a_{\beta_2}$</th>
<th>$a_{\beta_3}$</th>
<th>$a_{\beta_4}$</th>
<th>$a_{\beta_5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$t + 1$</td>
<td>$t + 2$</td>
<td>$t + 3$</td>
<td>$t + 4$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3.1. The DSNP P System as Generator

**Theorem 1.** $N_{\text{gen}}(\text{DSNP}^2) = \text{NRE}$.

**Proof.** In generating mode, all registers are empty in the initial state, starting with instruction $I_0$. 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 $a_{\beta_1}$ which has one spike. The module ADD is used to simulate ADD instruction $a_i$: $(\text{ADD}(r), I_j, I_k)$, shown in Figure 2. The reference neuron in this module is neuron $a_{\beta_1}$, denoted as $a_{\beta_1}$. Assume the rule in neuron $a_{\beta_1}$ is activated at time $t$, and it will send one spike to neurons $a_{\beta_1}$ and $a_{\beta_2}$, respectively. At time $t + 1$, neurons $a_{\beta_1}$ and $a_{\beta_2}$ receive this spike. Meanwhile, the rule in $a_{\beta_2}$ is nondeterministic chosen one to perform, so there are two cases:

1. At time $t + 1$, if the rule $a \rightarrow a; [1, 2]$ is used, the neuron $a_{\beta_1}$ will accept one spike from neuron $a_{\beta_1}$, at time $t + 2$, so that the rule $a \rightarrow a; [2, 3]$ in neuron $a_{\beta_1}$ can be activated. At the same time, neuron $a_{\beta_2}$ receives one $a$ from neuron $a_{\beta_1}$ at time $t + 2$, and the rule $a \rightarrow a; [1, 2, 3]$ in neuron $a_{\beta_1}$ is applied. At time $t + 3$, both $a_{\beta_1}$ and $a_{\beta_2}$ receive two spikes from neuron $a_{\beta_1}$, so the rule $a^2 \rightarrow a; [3, 4]$ can be used. Therefore, the neuron $a_{\beta_1}$ is empty, and neuron $a_r$ has two spikes.
(2) At time $t + 1$, if the rule $a \rightarrow \lambda; [1, 2]$ is applied, then the neuron $\sigma_i$ is empty at time $t + 2$, and the rule $a \rightarrow a; [1, 2, 3]$ in neuron $\sigma_i$ is enabled. Each of neurons $\sigma_2$ and $\sigma_i$ receives one spike from neuron $\sigma_3$ at time $t + 3$. So the rule $a \rightarrow a; [3, 4]$ in neuron $\sigma_2$ can perform, and then, at time $t + 4$, neurons $\sigma_3$ and $\sigma_i$ receive one spike, respectively. In this way, there are two spikes in the neuron $\sigma$, 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 $l_i$: (SUB($r, l_j, l_k$)). The simulation of module SUB also starts at reference neuron $\sigma_r$. The neuron $\sigma_r$ activates at time $t$ and sends one spike to neurons $\sigma_j$ and $\sigma_k$, respectively, before time $t + 1$. It means that neurons $\sigma_j$ and $\sigma_k$ receive this spike at time $t + 1$. Depending on whether the neuron $\sigma$ has spikes at the initial time, there are two cases as follows:

(1) If the neuron $\sigma$ is not empty, it has $2n$ spikes. So, neuron $\sigma$ has $2n + 1$ spikes (odd number) at time $t + 1$, and the rule $(a(aa)^+a^3) \rightarrow a; [1, 2]$ can be used. At time $t + 2$, both neurons $\sigma_3$ and $\sigma_4$ receive one spike from the neuron $\sigma_r$, and the rules $a \rightarrow \lambda; [2, 3]$ in neurons $\sigma_3$ and $\sigma_4$ can be used. At the same time, $\sigma_5$ also receives one spike from $\sigma_3$; that is, the rule $a \rightarrow a; [2, 3, 4]$ in neuron $\sigma_5$ can be used. At time $t + 3$, neurons $\sigma_3$ and $\sigma_5$ get one spike from neuron $\sigma_4$, such that rules $a \rightarrow \lambda; [3, 4]$ in $\sigma_3$ and $\sigma_5$ are performed. At the next time, each of neurons $\sigma_3$ and $\sigma_5$ receives one spike from $\sigma_r$, and rule $a \rightarrow a; [4, 5]$ in $\sigma_3$ and rule $a \rightarrow \lambda; [4, 5]$ in $\sigma_5$ are applied. In this way, neuron $\sigma_b$ is empty without receiving any spikes and $\sigma_i$ gains one spike.

(2) There is no spike in neuron $\sigma_r$. At time $t + 1$, rules $a \rightarrow a; [1, 2, 3]$ allow neurons $\sigma_2$ and $\sigma_3$ to obtain one spike from neuron $\sigma_r$, and this rule is activated at time $t + 2$. And then, each of neurons $\sigma_3$ and $\sigma_4$ obtains two spikes which come from $\sigma_3$ and $\sigma_i$, so the rule $a^2 \rightarrow \lambda; [3, 4]$ in $\sigma_3$ and the rule $a^2 \rightarrow a; [3, 4]$ in $\sigma_4$ can fire. Therefore, $\sigma_i$ receives one spike from $\sigma_3$, and neuron $\sigma_j$ 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 $l_i$: (SUB($r, l_j, l_k$)), and $l_j$: (SUB($r, l_j', l_k'$) are simulated. When neuron $\sigma_i$ fires, neurons $\sigma_{i_1}$ and $\sigma_{i_2}$ get one spike at time $t + 2$ or time $t + 3$. But only rules $a \rightarrow \lambda; [2, 3]$ and $a \rightarrow \lambda; [3, 4]$ in neurons $\sigma_{i_1}$ and $\sigma_{i_2}$ are enabled. Neurons $\sigma_{i_1}$ and $\sigma_{i_2}$ 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 $\sigma_{out}$ has one spike. Assume that the neuron $\sigma_1$ has $2n$ spikes according to the number $n$ storing in register 1 of $M$ and neuron $\sigma_{out}$ receives one spike at time $t$. So the neuron $\sigma_1$ gets one spike at time $t + 1$. Thus, the neuron $\sigma_1$ has $2n + 1$ spikes, which is odd number, and at time $t + 2$, the rule $a(aa)^+a^3 \rightarrow a$ is activated and sends a spike to neurons $\sigma_{i_1}$ and $\sigma_{out}$, respectively. In this way, $\sigma_{out}$ has two spikes, which is even number. The rule in $\sigma_{out}$ can be applied the next time and sends one spike to environment. At the same time, neuron $\sigma_{i_1}$ sends one spike to neuron $\sigma_{out}$ and neuron $\sigma_1$ fires again. At present, there are three spikes in neuron $\sigma_{out}$. No rules in $\sigma_{out}$ can be enabled since the number of spikes is odd.

After $n + 1$ times, that is, at time $t + n + 1$, neuron $\sigma_1$ has only one spike left, so that the rule in $\sigma_1$ can no longer be used. At this moment, neuron $\sigma_{i_1}$ has one spike. Therefore, at time $t + n + 2$, $\sigma_{i_1}$ is activated and sends a spike to $\sigma_{out}$. The number of spikes in $\sigma_{out}$ 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.
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 \(\Pi\) correctly simulates the register machine \(M\). Therefore, the conclusion of Theorem 1 is proved.

\[\square\]

### 3.2. The DSN P System as Acceptor

**Theorem 2.** \(N_{\text{acc}} \text{DSNP}^2 = \text{NRE}\).

**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 \(\sigma_r\), and if the number \(n\) is stored in register \(r\), then there are \(2n\) spikes in neuron \(\sigma_r\). If the register machine reaches the halting instruction \(I_h\), 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 \(\Pi\), all neurons are empty, except for reference neuron \(\sigma_0\) which contains one spike. Module ADD' shown in Figure 6 differs from model ADD shown in Figure 2. Module ADD' simulates the ADD' instruction \(I_1\): \((\text{ADD}(r), I_h)\). Assume that neuron \(\sigma_{i'}\) has one spike and activates at time \(t\). Thus, each of neurons \(\sigma_r\) and \(\sigma_m\) receives one spike from \(\sigma_i\) at time \(t + 1\). And the rule in neuron \(\sigma_m\) can be applied. At time \(t + 2\), both neurons \(\sigma_r\) and \(\sigma_i\) receive a spike, and at this time, neuron \(\sigma_r\) has two spikes. The simulation of module ADD' is complete, since there are two spikes in \(\sigma_r\) and the content of register \(r\) increases one.

Module INPUT, shown in Figure 7, is used to compute number \(n\) introduced by neuron \(\sigma_m\) into \(\sigma_1\). The module INPUT computes number \(n\) between the time of receiving two spikes and the function of this module is to read the spike train \(10^{n-1}\), where \(1 \leq n \in \mathbb{N}^+\). Assume the neuron \(\sigma_n\) receives a spike at time \(t\). Each of neurons \(\sigma_{j_1}, \sigma_{j_2},\) and \(\sigma_{j_3}\) receives one spike at time \(t + 1\). At the next time, rules in neurons \(\sigma_{j_1}\) and \(\sigma_{j_2}\) are enabled; thus, one spike is received by both neurons \(\sigma_{j_1}\) and \(\sigma_1\) which comes from the neuron \(\sigma_{j_2}\). And at the same time, each of neurons \(\sigma_{j_2}\) and \(\sigma_{j_3}\) receives a spike from neuron \(\sigma_{j_1}\). Therefore, neuron \(\sigma_1\) has two spikes at that moment. Notably, rules in neurons \(\sigma_{j_1}\) and \(\sigma_{j_2}\) do not stop until \(\sigma_m\) received the second spike. After

![Figure 4: Module SUB-SUB: simulating the SUB instructions](image)

![Figure 5: Module OUTPUT](image)

### Table 2: The results of module OUTPUT

<table>
<thead>
<tr>
<th></th>
<th>(\sigma_{i_1})</th>
<th>(\sigma_1)</th>
<th>(\sigma_{d_1})</th>
<th>(\sigma_{\text{out}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t)</td>
<td>(t + 1)</td>
<td>(2n)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(t + 1)</td>
<td>0</td>
<td>(2n + 1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(t + 2)</td>
<td>0</td>
<td>(2(n - 1) + 1)</td>
<td>1</td>
<td>2 (fire)</td>
</tr>
<tr>
<td>(t + 3)</td>
<td>0</td>
<td>(2(n - 2) + 1)</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(t + 4)</td>
<td>0</td>
<td>(2(n - 3) + 1)</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td></td>
</tr>
<tr>
<td>(t + n + 1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2n - 1</td>
</tr>
<tr>
<td>(t + n + 2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2n (fire)</td>
</tr>
</tbody>
</table>
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 $f: N^K \rightarrow N$ 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 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:

$$ l_{22}: \text{(SUB}(0, l_{23}, l_{h''}), $$

$$ l_{23}: \text{(ADD}(8, l_{22})) $$

(2)

$$ l_{h''}: \text{HALT}. $$

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 $l_j: (\text{ADD}(r), l_j)$ 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.
In the beginning, the two parameters \( g(x) \) and \( y \) are encoded into the form of a spike train \( 10^{g(x)-1}10^{y-1} \). Then, through the computing of module INPUT, the two parameters \( g(x) \) and \( y \) are stored in neurons \( \sigma_1 \) and \( \sigma_2 \) replaced by \( 2g(x) \) spikes and \( 2y \) spikes, respectively.

Module INPUT shown in Figure 10 is used to compute the function \( 10^{g(x)-1}10^{y-1} \) that is introduced by neuron \( \sigma_m \) into \( \sigma_1 \) with \( 2g(x) \) spikes and \( 2y \) spikes. Assume that the neuron \( \sigma_m \) activates at time \( t \). Each of neurons \( \sigma_{m_1} \), \( \sigma_{m_2} \), \( \sigma_{m_3} \), \( \sigma_{m_4} \), \( \sigma_{m_5} \), and \( \sigma_{m_6} \) receives one \( a \) at time \( t + 1 \). Thus, neurons \( \sigma_{m_1} \) and \( \sigma_{m_2} \) activate, and then neuron \( \sigma_{m_3} \) sends one spike to \( \sigma_1 \) and \( \sigma_{m_4} \), respectively. Meanwhile, neuron \( \sigma_{m_5} \) sends one spike to \( \sigma_1 \) and \( \sigma_{m_6} \) so that neuron \( \sigma_1 \) receives two spikes at time \( t + 2 \), and each of neurons \( \sigma_{m_3} \) and \( \sigma_{m_4} \) has one spike. Until the neuron \( \sigma_{m_3} \) receives the second spike, that is, after \( g(x) - 1 \) times, rules in \( \sigma_{m_3} \) and \( \sigma_{m_4} \) cannot fire. In this way, neuron \( \sigma_1 \) gets \( 2g(x) \) spikes corresponding to the number \( g(x) \) stored in register 1.

Simultaneously, each of neurons \( \sigma_{m_1}, \sigma_{m_2}, \) and \( \sigma_{m_3} \) has two spikes when neuron \( \sigma_1 \) gets \( 2g(x) \) spikes. Thus, the rules in \( \sigma_{m_4} \) and \( \sigma_{m_5} \) are enabled, both consuming one spike. And at the next time, neuron \( \sigma_2 \) receives two spikes, one from \( \sigma_{m_2} \) and the other from \( \sigma_{m_5} \). At the same time, neurons \( \sigma_{m_3} \) and \( \sigma_{m_6} \) each receive a spike from the other, so the number of spikes they contain returns to an even number. In this way, neurons \( \sigma_{m_4} \) and \( \sigma_{m_5} \) remain active for the next \( y - 1 \) times. Therefore, neuron \( \sigma_2 \) accepts \( 2y \) spikes altogether until neuron \( \sigma_m \) receives the third spike. And then, neuron \( \sigma_{m_3} \) has three spike, and the rule \( a(r_s) \rightarrow a \) can perform and sends one spike to \( \sigma_{m} \).

Therefore, the system needs 94 neurons to simulate the universal register machine \( M_u \) including

(i) 6 neurons for module INPUT
(ii) \( 1 \times 10 \) neurons for 10 ADD instructions
(iii) \( 3 \times 14 \) 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 \( I_1 \) : (SUB (r_1), r_2, r_3) and ADD instructions \( I_2 \) : (ADD (r_2), r_3), as shown in Figure 11, where \( r_1 \neq r_2 \). The simulation of module SUB-ADD-1 also starts at reference neuron \( \sigma_r \). When neuron \( \sigma_r \) has one spike, ADD instruction is triggered and performed immediately. Thus, the final results should be that neuron \( \sigma_r \) gains one spike, the register \( r_1 \) increases by one, and \( \sigma_r \) is empty. If neuron \( \sigma_r \) has one spike, the ADD instruction will not activate. The register \( r_2 \) and neuron \( \sigma_r \) are certainly empty. The computation process is divided into two cases depending on the number of spikes in neuron \( \sigma_r \). The process is similar to module SUB (shown in Figure 3).

Assume that the reference neuron \( \sigma_r \) fires at time \( t \).

(1) If the register \( r_1 \) is nonempty, neuron \( \sigma_{r_1} \) has \( 2n \) spikes.

At time \( t + 1 \), neuron \( \sigma_1 \) has \( 2n + 1 \) spikes, and the rule \( a(aa)^+ \rightarrow a \) : (1, 2) fires. At the same time, neuron \( \sigma_{c_1} \) also gains one spike. Thus, at the next time, each of neurons \( \sigma_1, \sigma_{c_1}, \) and \( \sigma_{c_2} \) gains one spike, and the rule \( a \rightarrow \lambda ; (2, 3) \) in neuron \( \sigma_{c_2} \) and \( a \rightarrow \lambda ; (2, 3) \)
In $\sigma_a$, can be used. At time $t + 3$, neurons $\sigma_a$ and $\sigma_c$, gain one spike form $\sigma_a$, and rule $a \rightarrow \lambda; [3, 4]$ can be applied. And at the next time, neurons $\sigma_c$ and $\sigma_c$ get one spike form $\sigma_c$, at again, thus, the rule $a \rightarrow a; [4, 5]$ in $\sigma_c$, and $a \rightarrow \lambda; [4, 5]$ in $\sigma_c$, fire. At time $t + 5$, $\sigma_c$, applies its rule $a \rightarrow a; [5, 6]$ and sends one spike to $\sigma_a$ before time $t + 6$. In this way, $\sigma_r$ has two spikes, and neuron $\sigma_{r_j}$ is empty.

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

In this way, consecutive SUB-ADD instructions can be simulated correctly, and neuron $\sigma_{r_j}$ 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{align*}
l_0: & (\text{SUB}(1), l_1, l_2), \\
l_1: & (\text{ADD}(7), l_0), \\
l_4: & (\text{SUB}(6), l_3, l_4), \\
l_5: & (\text{ADD}(5), l_6), \\
l_8: & (\text{SUB}(7), l_7, l_8), \\
l_9: & (\text{ADD}(1), l_4), \\
l_{16}: & (\text{SUB}(6), l_9, l_{16}), \\
l_{17}: & (\text{ADD}(6), l_{10}), \\
l_{18}: & (\text{SUB}(5), l_{16}, l_{17}), \\
l_{19}: & (\text{ADD}(4), l_{11}), \\
l_{22}: & (\text{SUB}(1), l_{23}, l_{24}), \\
l_{23}: & (\text{ADD}(8), l_{22}).
\end{align*}
Module SUB-ADD-2 is used to simulate the combination of SUB instruction $l_{15}$: $(\text{SUB}(3), l_{18}, l_{20})$ and ADD instruction $l_{20}$: $(\text{ADD}(0), l_0)$. 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 instruction $l_{18}$: $(\text{ADD}(r_1), l_g)$ and SUB instruction $l_g$: $(\text{SUB}(r_2), l_j, l_k)$.

Figure 14: Module ADD-ADD: simulating ADD-ADD instructions $l_{17}$: $(\text{ADD}(2), l_{11})$ and $l_{21}$: $(\text{ADD}(3), l_{18})$. 

Module SUB-ADD-2 is used to simulate the combination of SUB instruction $l_{15}$: $(\text{SUB}(3), l_{18}, l_{20})$ and ADD instruction $l_{20}$: $(\text{ADD}(0), l_0)$. 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.
instructions \( l_1: (\text{ADD}(r_1), l_9) \) and \( l_9: (\text{SUB}(r_2), l_7, l_k) \), where \( r_1 \neq r_2 \). Assume that reference neuron \( \sigma_r \) has one spike at time \( t \), and the produced spike is received by \( \sigma_{r_1} \) and \( \sigma_{r_2} \) at the next time. Then, the working process of next time is the same as module \( \text{SUB} \), divided into two cases. \( \sigma_{r_1} \) receives one spike from \( \sigma_r \) at time \( t + 2 \) or time \( t + 3 \), so that \( \sigma_{r_1} \) has two spikes. Thus, both the \( \text{ADD} \) and \( \text{SUB} \) instructions can be implemented correctly by this module. So, neuron \( \sigma_{l_1} \) can be saved, and neuron \( \sigma_{m_i} \) also is saved as compared to module \( \text{ADD} \) shown in Figure 6. There are two pairs of instructions that adapt to the module \( \text{ADD-SUB} \). Thus, four neurons are saved.

\[
\begin{align*}
  l_5: & \quad (\text{ADD}(5), l_6), \\
  l_6: & \quad (\text{SUB}(7), l_7, l_k), \\
  l_9: & \quad (\text{ADD}(6), l_{10}), \\
  l_{10}: & \quad (\text{SUB}(4), l_9, l_{11}).
\end{align*}
\]

ADD-ADD instructions \( l_1: \text{(ADD}(2), l_{12}) \) and \( l_{12}: \text{(ADD}(3), l_{18}) \) can also be combined by module ADD-ADD (shown in Figure 14). Assume that, at time \( t \), neuron \( \sigma_{l_1} \) activates. Each of neurons \( \sigma_{m_5}, \sigma_{m_6}, \) and \( \sigma_2 \) receives one spike at time \( t + 1 \). Then, neurons \( \sigma_3 \) and \( \sigma_2 \) get their second spike from \( \sigma_{m_i} \) 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 \( \sigma_{m_i} \), and the neuron associated with label \( l_{12} \) is also saved. Therefore, two neurons can be saved by module ADD-ADD.

From the above proof, six neurons are saved by module \( \text{SUB-ADD-1} \). One neuron is saved by module \( \text{SUB-ADD-2} \). Four neurons are saved by module \( \text{ADD-SUB} \). Two neurons are saved by module \( \text{ADD-ADD} \). 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 \( \text{SN P} \) systems, called \( \text{DSN P} \) 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 \( \text{SN P} \) 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, \( \text{DSN P} \) 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 \( \text{DSN P} \) systems are proved by simulating the register machine under the generating mode and the accepting mode. In Theorem 3, a universal \( \text{DSN P} \) system having 81 neurons for computing functions is also proved.

In this paper, we only involve the theoretical proof of \( \text{DSN P} \) systems. In the future work, both theory and application, there are still many problems that can be improved based on \( \text{DSN P} \) systems. In theory, we can further improve \( \text{DSN P} \) 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 \( \text{DSN P} \) 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 \( \text{DSN P} \) system can be combined with a neural network and applied to image or text processing, as the work in [50]. On the other hand, \( \text{DSN P} \) 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 \( \text{SN P} \) systems has not been fully studied, as a third-generation neural network, the development of \( \text{SN P} \) 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


