Dynamic Threshold Neural P Systems with Multiple Channels and Inhibitory Rules

Authors:

Xiu Yin, Xiyu Liu

Date Submitted: 2021-04-29

Keywords: multiple channels, inhibitory rules, dynamic threshold neural P systems, spiking neural P systems, membrane computing

Abstract:

In biological neural networks, neurons transmit chemical signals through synapses, and there are multiple ion channels during transmission. Moreover, synapses are divided into inhibitory synapses and excitatory synapses. The firing mechanism of previous spiking neural P (SNP) systems and their variants is basically the same as excitatory synapses, but the function of inhibitory synapses is rarely reflected in these systems. In order to more fully simulate the characteristics of neurons communicating through synapses, this paper proposes a dynamic threshold neural P system with inhibitory rules and multiple channels (DTNP-MCIR systems). DTNP-MCIR systems represent a distributed parallel computing model. We prove that DTNP-MCIR systems are Turing universal as number generating/accepting devices. In addition, we design a small universal DTNP-MCIR system with 73 neurons as function computing devices.

Record Type: Published Article

Submitted To: LAPSE (Living Archive for Process Systems Engineering)

Citation (overall record, always the latest version):	LAPSE:2021.0275
Citation (this specific file, latest version):	LAPSE:2021.0275-1
Citation (this specific file, this version):	LAPSE:2021.0275-1v1

DOI of Published Version: https://doi.org/10.3390/pr8101281

processes

MDPI

Article

Dynamic Threshold Neural P Systems with Multiple Channels and Inhibitory Rules

Xiu Yin and Xiyu Liu *

Business School, Shandong Normal University, Jinan 250358, China; xiuyinsdnu@163.com
* Correspondence: xyliu@sdnu.edu.cn

Received: 29 September 2020; Accepted: 9 October 2020; Published: 13 October 2020

Abstract: In biological neural networks, neurons transmit chemical signals through synapses, and there are multiple ion channels during transmission. Moreover, synapses are divided into inhibitory synapses and excitatory synapses. The firing mechanism of previous spiking neural P (SNP) systems and their variants is basically the same as excitatory synapses, but the function of inhibitory synapses is rarely reflected in these systems. In order to more fully simulate the characteristics of neurons communicating through synapses, this paper proposes a dynamic threshold neural P system with inhibitory rules and multiple channels (DTNP-MCIR systems). DTNP-MCIR systems represent a distributed parallel computing model. We prove that DTNP-MCIR systems are Turing universal as number generating/accepting devices. In addition, we design a small universal DTNP-MCIR system with 73 neurons as function computing devices.

Keywords: membrane computing; spiking neural P systems; dynamic threshold neural P systems; inhibitory rules; multiple channels

1. Introduction

Membrane computing (MC) is a type of system with the characteristic of distributed parallel computing, usually called P systems or membrane systems. MC is obtained by researching the structure and functioning of biological cells as well as the communication and cooperation of cells in tissues, organs, and biological neural networks [1,2]. P systems are mainly divided into three categories, namely cell-like P systems, tissue-like P systems, and neural-like P systems. In the past two decades, many P-system variants have been studied and applied to real-world problems, and most of them have been proven to be universal number generating/accepting devices and functional computing devices [3,4].

1.1. Related Work

Spiking neural P (SNP) systems are abstracted from biological facts that neurons transmit spikes to each other through synapses. An SNP system can be regarded as a directed graph, where neurons are regarded as nodes, and the synaptic connections between neurons are regarded as arcs [5,6]. SNP systems are the main form of neural-like P systems [7]. An SNP system consists of two components: data and firing rules. Data usually describe the states of neurons and can also indicate the number of spikes contained in every neuron. Firing rules contain spiking rules and/or forgetting rules. The firing of rules needs to satisfy necessary conditions. The form of firing rules is $E/a^c \rightarrow a^p$, where E indicates the regular expression. When $a^n \in L(E)$ and $n \geq c$, the rule is applied and the neuron fires. $L(E)$ represents the language set associated with regular expressions E. a^n indicates that the neuron where the rule is located contains n spikes. Once the rule is enabled, the neuron removes c spikes and transmits the generated p spikes to succeeding neurons. When $p = 0$, rule $E/a^c \rightarrow \lambda$ is called a forgetting rule. λ represents an empty string, which indicates that forgetting rules does not generate

new spikes. In summary, the firing condition of a firing rule is: $a^n \in L(E)$ and $n \geq c$. This means that the firing of a rule is only related to the state of the neuron and has nothing to do with the state of other neurons. Moreover, the parallel work of neurons determines that SNP systems are models of distributed parallel computing. Therefore, most SNP systems are equipped with a global clock to mark time. SNP systems also have nondeterminism characteristic. If more than one rule can be enabled in a neuron at a certain time, then only one of them will be selected non-deterministically.

Neural-like P systems have received extensive attention and research in recent years. SNP systems was proposed by Ionescu et al. [7]. Usually, SNP systems have spikes and forgetting rules, but Song et al. [8] proposed that neurons can also have request rules. The request rules enable neurons to perceive "stimuli" from the environment by receiving a certain number of spikes. Zeng et al. [9] proposed SNP systems with weights. In this system, when the potential of a neuron is equal to a given value (called a threshold), it will fire. Zhao et al. [10] introduced a new mechanism called neuron dissolution, which can eliminate redundant neurons generated in the calculation process. In order to enable SNP systems to represent and process fuzzy and uncertain knowledge, the weighted fuzzy peak neural P system was proposed by Wang et al. [11]. Considering that SNP systems can only fire one neuron in each step, sequential SNP systems is investigated [12,13]. Although most SNP systems are synchronous, asynchronous SNP systems are investigated by [14–16]. Song et al. [15] also studied the computing power of asynchronous SNP system with partial synchronization. Moreover, most SNP systems as number generating/receiving devices [17,18], language generators [19], and functional computing devices [20,21] have been proven to be Turing universal. Furthermore, SNP systems have applications in dealing with real-world problems, such as fault diagnosis [22–24], clustering [25], and optimization problems [26].

1.2. Motivation

Yang et al. [27] researched SNP systems with multiple channels (SNP-MC systems) based on the fact: there are multiple ion channels in the process of chemical signal transmission, therefore, spiking rules with channel labels are introduced into neural P systems. After SNP-MC systems were proposed, many neural P systems combine with multiple channels have been investigated. They prove that using multiple channels can improve the computing power of neural P systems.

In addition, there are the following facts in the biological nervous system:

(1) The conduction of nerve impulses between neurons is unidirectional, that is, nerve impulses can only be transmitted from the axon of one neuron to the cell body or dendrites of another neuron, but not in the opposite direction.

(2) Synapses are divided into excitatory synapses and inhibitory synapses. According to the signal from the presynaptic cells, if the excitability of the postsynaptic cell is increased or excited, then the connection is excitatory synapse. If the excitability of the postsynaptic cell is decreased or the excitability is not easily generated, then it is inhibitory synapse.

Li et al. [28] were inspired by the above two biological facts and proposed SNP system with inhibitory rules (SNP-IR systems). The firing condition of an inhibitory rule not only related to the state of neuron where the rule is located but also related to other neurons (presynaptic neurons). This is because of the unidirectionality of nerve impulse transmission. SNP-IR systems have stronger control capabilities than other SNP systems.

Peng et al. [29] first proposed the dynamic threshold neural P systems (DTNP systems), which were abstracted from ICM model of the cortical neurons. The firing mechanism of DTNP systems are different from SNP systems. DTNP systems adopt a dynamic-threshold-based firing mechanism, and they have two data units (feeding input unit and dynamic threshold unit) as well as a maximum spike consumption strategy. In order to improve the computational efficiency of DTNP systems, we introduce multiple channels and inhibitory rules.

The main motivation of this paper is mainly to introduce inhibitory rules into DTNP systems, which can better reflect the working mechanism of inhibitory synapses and simulate the actual situation of communication between neurons through synapses. The introduction of multiple channels improves the control capability of the DTNP system so that it can better solve real-world problems.

Compared with DTNP systems, DTNP-MCIR systems have the following innovations:

(1) DTNP-MCIR systems introduce firing rules with channel labels.
(2) Inspired by SNP-IR systems, we introduce inhibitory rules to DTNP systems, but the form and firing conditions of inhibitory rules have been re-defined.
(3) The firing rules of neuron σ_r (corresponds to register r) is $E_r / (a^u, a^\tau) \rightarrow a^p(l)$ in DTNP systems, E_r is a default firing condition usually not displayed, because $E_r = \{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau\}$ can be reflected in the rules. However, E_r can also be a regular expression in DTNP-MCIR systems, for example $a^{2b+1} / (a^u, a^\tau) \rightarrow a^p(1)$, where a^{2b+1}, $b \geq 1$ represents an odd number of spikes. The rule is only enabled when neuron σ_r contains an odd number of spikes and satisfies the default firing condition.

If a rule in a neuron is enabled in DTNP-MCIR systems, then the number of spikes consumed by the neuron is $u + \tau - p - n$. When two rules in a neuron can be applied simultaneously at time t, we can choose one of the rules according to the neuron's maximum spike consumption strategy.

Based on the above, we built a variant of DTNP systems called dynamic threshold neural P systems with multiple channels and inhibitory rules (DTNP-MCIR systems). We will prove Turing universal of DTNP-MCIR systems as number generating/accepting devices and functional computing devices. The content of the rest of this paper is arranged as follows. Section 2 defines DTNP-MCIR systems and gives an illustrative example. Section 3 illustrates the Turing universality of DTNP-MCIR systems as number-generating/accepting devices. Section 4 explains DTNP-MCIR systems as function computing devices. Section 5 is the conclusion of this paper.

2. DTNP-MCIR Systems

In this section, we define a DTNP-MCIR system, describe some of its related details, and give an illustrative example. For convenience, DTNP-MCIR systems use the same notations and terms as SNP systems.

2.1. Definition

A DTNP-MCIR system Π with degree $m \geq 1$ is shown below:

$$\Pi = (O, L, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$$

where:

(1) $O = \{a\}$ is a singleton alphabet (the object a is called the spike);
(2) $L = \{1, 2, \ldots, N\}$ is the channel labels;
(3) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are m neurons of the form $\sigma_i = (u_i, \tau_i, L_i, R_i), 1 \leq i \leq m$ where:
    (a) $u_i \geq 0$ is the number of spikes contained in the feeding input unit of neuron σ_i;
    (b) $\tau_i \geq 0$ is the number of spikes as a (dynamic) threshold in neuron σ_i;
    (c) $L_i \subseteq L$ is a finite set of channel labels used by neuron σ_i;
    (d) R_i is a finite set of rules of two types in σ_i;
        (i) the form of firing rules is $E_i / (a^u, a^\tau) \rightarrow a^p(l)$, where $E_i = \{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau\}$ is a firing condition, $u \geq 1, \tau \geq 0, p \geq 0$, $p \leq u$ and $l \in L_i$; when $p \geq 1$, the rule is known as a spiking rule, and when $p = 0$, $a^0 = \lambda$, the rule is known as a forgetting rule, can be written as $E_i / (a^u, a^\tau) \rightarrow \lambda$;

(ii) the form of inhibitory rules is $E_{(i,j)}/\left(a^u, \overline{a^\tau}\right) \rightarrow a^p(1)$, where the subscript (i,j) represents an inhibitory arc between neuron σ_i and σ_j, $E_{(i,j)} = \left\{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau \cap u_j \neq \tau_i\right\}$, $u \geq 1, \tau \geq 0, p \geq 0$, $p \leq u$, $l \in L_i$, and $1 \leq j \leq m, i \neq j$;

(4) $syn = (i,j,l) \subseteq \{1,2,\ldots,m\} \times \{1,2,\ldots,m\} \times L$ with $i \neq j$ (synapse connections)

(5) in indicate input neurons;

(6) out indicate output neurons.

From the perspective of topological structure, a DTNP-MCIR system can be regarded as a directed graph with inhibitory arcs, where m neurons denote m nodes, and the synaptic connections between neurons denote the arcs of the directed graph. An inhibitory arc is represented by an arc with a solid circle. A directed arc with an arrow indicates excitatory conduction. Synaptic connections are expressed as a tuple $(i,j,l) \in syn$, which means that neuron σ_i is connected to neuron σ_j through the l-th channel. Outside system Π is called environment. In system Π, only input neurons σ_{in} and output neurons σ_{out} can communicate with the environment. When the system Π works in generative mode, input neurons σ_{in} are not considered, by contrast, output neurons σ_{out} are not considered in accepting mode.

Data and rules are two components of a DTNP-MCIR system. There are two data units in each neuron σ_i: feeding input unit u_i and dynamic threshold unit τ_i. If the calculation of system Π proceeds to time t, then $u_i(t)$ and $\tau_i(t)$ respectively represent the number of spikes in the two data units of the neuron σ_i [29]. The direct manifestation of data is the number of spikes contained in each neuron, the number of spikes contained in a neuron can also indicate the state of the neuron. The state of the entire system at time t is characterized by the configuration $C_t = ([u_1(t), \tau_1(t)], [u_2(t), \tau_2(t)], \ldots, [u_m(t), \tau_m(t)])$, which can reflect the number of spikes in feeding input unit and dynamic threshold unit of m neurons. The initial configuration can be denoted as $C_0 = ([u_1(0), \tau_1(0)], [u_2(0), \tau_2(0)], \ldots, [u_m(0), \tau_m(0)])$.

The rules in DTNP-MCIR systems can be divided into two types: firing rules and inhibitory rules. The form of firing rules is $E_i/(a^u, a^\tau) \rightarrow a^p(l)$, firing condition is $E_i = \{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau\}$ which is equivalent to $u_i(t) \geq \tau_i(t)$, $u_i(t) \geq u$ and $\tau_i(t) \geq \tau$. If the firing condition in neuron σ_i is fulfilled at time t, then rules in neuron σ_i will be applied to a configuration C_t. When rule $E_i/(a^u, a^\tau) \rightarrow a^p(l)$ is enabled, neuron σ_i fires and then consumes u spikes from feeding input unit (retaining ($u_i(t) - u$) spikes) and τ spikes from dynamic threshold unit (retaining ($\tau_i(t) - \tau$)spikes). When $p \geq 1$, p spikes generated by neuron σ_i are transmitted to neuron σ_j through the l-th synaptic channel. When $p = 0$, rule $E_i/(a^u, a^\tau) \rightarrow \lambda$ is called forgetting rule. If forgetting rule in neuron σ_i is enabled, then spikes in feeding input unit and dynamic threshold unit will be removed and no new spikes are generated.

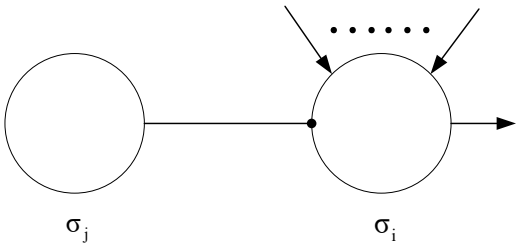
The form of inhibitory rules is $E_{(i,j)}/\left(a^u, \overline{a^\tau}\right) \rightarrow a^p(l)$, as shown in Figure 1a, its firing mechanism is special. Usually the firing of a neuron depends only on its current state, other neurons have no influence on it. However, the firing condition of an inhibitory rule not only depends on the state of the current neuron, but also related to the state of the preceding neuron (called an inhibitory neuron). Suppose an inhibitory rule is located in neuron σ_i and the inhibitory neuron of σ_i is σ_j. We assume that when there is an inhibitory arc between neurons σ_i and σ_j, the usual arc cannot exist. In addition, the inhibitory neuron σ_j only controls the firing of the neuron σ_i, and the neuron σ_i has no effect on the inhibitory neuron σ_j. The firing condition of the inhibitory rule is defined as $E_{(i,j)} = \left\{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau \wedge u_j = \tau_i\right\}$, which is equivalent to $u_i(t) \geq \tau_i(t)$, $u_i(t) \geq u$, $\tau_i(t) \geq \tau$ and $u_j(t) = \tau_i(t)$,where $u_j(t) = \tau_i(t)$ is an additional constraint compared to the firing condition of the firing rule. $u_j(t)$ represents the number of spikes at time t in the feeding input unit of inhibitory neuron σ_j. $\tau_i(t)$ represents the number of spikes at time t in the dynamic threshold unit of neuron σ_i. Therefore, the firing conditions of inhibitory rules correspond to the function of inhibitory synapses. It is not only related to the state of neuron σ_i, but also depends on the state of the inhibitory neuron σ_j. Although the firing conditions of inhibitory rules and firing rules are different, the spike

consumption strategy is same when they are applied. If neuron σ_i applies rules to configuration $C_t = ([u_1(t), \tau_1(t)], [u_2(t), \tau_2(t)], \ldots, [u_m(t), \tau_m(t)])$, we can get:

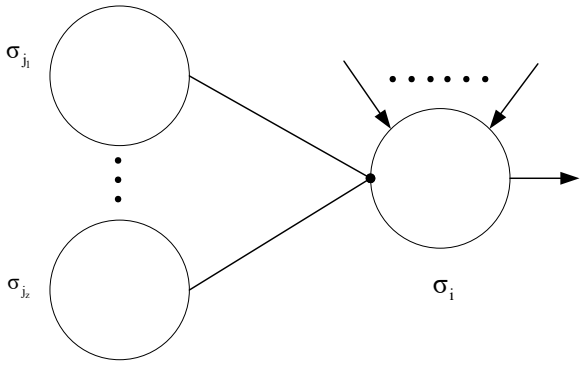
$$u_i(t+1) = \begin{cases} u_i(t) - u + n, \text{ if neuron } \sigma_i \text{ fires} \\ u_i(t) + n, \text{ otherwise} \end{cases} \quad (1)$$

$$\tau_i(t+1) = \begin{cases} \tau_i(t) - \tau + p, \text{ if neuron fires} \\ \tau_i(t), \text{ otherwise} \end{cases} \quad (2)$$

where n spikes come from other neurons, and p spikes are generated by the neuron σ_i.



(a) inhibitory rule : $E_{(i,j)} / \left(a^u, \overline{a^\tau}\right) \rightarrow a^p(1)$



(b) extended inhibitory rule : $E_{(i,j_1),\ldots,(i,j_z)} / \left(a^u, \overline{a^\tau}\right) \rightarrow a^p(1)$

Figure 1. (**a**) inhibitory rule and (**b**) extended inhibitory rule.

A neuron may have multiple inhibitory neurons. As shown in Figure 1b, neuron σ_i has z inhibitory neurons $\sigma_{j_1}, \ldots, \sigma_{j_z}$. At this time, the form of inhibitory rules is $E_{(i,j_1),\ldots,(i,j_z)} / \left(a^u, \overline{a^\tau}\right) \rightarrow a^p(1)$, which is called an extended inhibitory rule. The firing condition of the extended inhibitory rule is $u_i(t) \geq \tau_i(t), u_i(t) \geq u, \tau_i(t) \geq \tau$, and $u_{j_1}(t) = \tau_i(t) \wedge \ldots \wedge u_{j_z}(t) = \tau_i(t)$.

If neuron σ_i meets firing conditions during calculation, then one of the rules in R_i must be used. If two rules $E_i / (a^u, a^\tau) \rightarrow a^p(1)$ and $E_{i'} / \left(a^{u'}, a^{\tau'}\right) \rightarrow a^{p'}(1)$ in neuro n σ_i can be applied to configuration C_t at the same time, only one of them can be applied. According to the maximum spikes consumption strategy of DTNP-MCIR systems, when $u + \tau - p - n_1 > u' + \tau' - p' - n_2$, the rule $E_i / (a^u, a^\tau) \rightarrow a^p(1)$ is applied, otherwise, the rule $E_{i'} / \left(a^{u'}, a^{\tau'}\right) \rightarrow a^{p'}(1)$ is applied, when $u + \tau - p - n_1 = u' + \tau' - p' - n_2$, one of them will be non-deterministically selected. Note that this strategy is also effective for forgetting rules. For example, two rules $E_1 / \left(a^2, a\right) \rightarrow \lambda$ and $E_2 / (a, a) \rightarrow a(1)$ both satisfy firing conditions at time t, since the firing of forgetting rule $E_1 / \left(a^2, a\right) \rightarrow \lambda$ consumes three spikes, however, the spiking rule $E_2 / (a, a) \rightarrow a(1)$ only consumes one spike, then the forgetting rule will be chose and applied.

We assume that a transition step is from one configuration to another. A sequence of transitions from the initial configuration is defined as a calculation. For a configuration C_t, if no rules in the

system can be applied, the system calculation halts. Any calculation corresponds to a binary sequence, write 1 when the output neuron σ_{out} emit a spike to the environment, otherwise write 0. Therefore, the calculation result is defined as the time interval between the first two spikes emitted by the output neuron σ_{out}. $N_2(\Pi)$ represents a set of numbers calculated by system Π. $N_2DTNP - MCIR_m^n$ denotes the families of all sets $N_2(\Pi)$ accepted by DTNP-MCIR systems having at most m neurons and at most n rules in each neuron. When m or n is not restricted, the symbol "*" is usually used instead. System Π can be used as a accepting device, at this time, the input neuron receives spikes from the environment, but the output neuron is removed from the system. The system imports spikes train from the environment, then stores the number n in the form of 2n spikes. When the system calculation halts, n is the number to be accepted by it. $N_{acc}(\Pi)$ represents a set of numbers accepted by the system, $N_{acc}DTNP - MCIR_m^n$ denotes the families of all sets $N_{acc}(\Pi)$ accepted by DTNP-MCIR systems having at most m neurons and at most n rules in each neuron.

2.2. Illustrative Example

In order to clearly understand the working mechanism of DTNP-MCIR systems, we give an example that can generate a finite spike train, as shown in Figure 2.

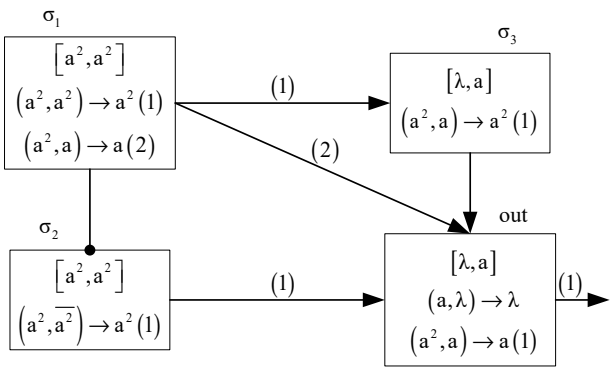


Figure 2. An illustrative example.

We assume that a DTNP-MCIR system Π consists of four neurons $\sigma_1, \sigma_2, \sigma_3, out$. Initially, the feeding input units of neuron σ_1 and neuron σ_2 each have two spikes, but there are no spikes in neuron σ_3 and out, that is $u_1(0) = 2, u_2(0) = 2, u_3(0) = 0, u_{out}(0) = 0$. The number of spikes in the initial dynamic threshold unit of the four neurons is $\tau_1(0) = 2, \tau_2(0) = 2, \tau_3(0) = 1, \tau_{out}(0) = 1$ respectively. Therefore, the initial configuration $C_0 = ([2,2], [2,2][0,1], [0,1])$.

At time 1, since $u_1(0) = 2 \geq \tau_1(0) = 2$, rules $(a^2, a^2) \to a^2(1)(u = \tau = p = 2)$ and $(a^2, a) \to a(2)(u' = 2, \tau' = p' = 1)$ can be applied in neuron σ_1, and since the number of spikes consumption for both rules is $u + \tau - p = u' + \tau' - p' = 2$, one of them will be selected non-deterministically. Therefore, there are the following two cases:

(1) Case 1: if rule $(a^2, a^2) \to a^2(1)$ is applied in neuron σ_1 at time 1, then neuron σ_1 will consume two spikes in the feeding input unit and sends two spikes to the neuron σ_3 through channel (1). Neuron σ_1 is the inhibitory neuron of neuron σ_2, since $u_2(0) = 2 \geq \tau_2(0) = 2$ and $u_1(0) = \tau_2(0)$, rule $(a^2, \overline{a^2}) \to a^2(1)$ reaches the firing condition in neuron σ_2, so neuron σ_2 consumes two spikes in the feeding input unit and sends two spikes to the neuron out via channel (1) at time 1. Therefore, $C_1 = ([0,2], [0,2][2,1], [2,1])$. At time 2, since $u_{out}(1) = 2 \geq \tau_{out}(1) = 1$, both rule $(a^2, a) \to a(1)$ and rule $(a, \lambda) \to \lambda(2)$ in neuron out can be applied. However, according to the maximum spike consumption strategy, rule $(a^2, a) \to a(1)$ will be applied and sends one spike to the environment through channel (1). Further, since $u_3(1) = 2 \geq \tau_3(1) = 1$, rule $(a^2, a) \to a(1)$ is enable in neuron σ_3 and consumes two spikes in the feeding input unit and sends two spike to neuron out. Thus $C_2 = ([0,2], [0,2][0,2], [2,1])$. At time 3, since $u_{out}(2) = 2 \geq \tau_{out}(2) = 1$, rule

$(a^2, a) \to a(1)$ fires again and sends one spike to the environment. The system Π halts. Therefore, the spike train generated by this system is "011".

(2) Case 2: if rule $(a^2, a) \to a^2(2)$ is applied in neuron σ_1 at time 1, then neuron σ_1 consumes two spikes in the feeding input unit and sends one spikes to the neuron out through channel (2). Because the state of system Π is the same as that in case 1 at time 1, neuron σ_2 consumes two spikes in the feeding input unit and sends two spikes to the neuron out through channel (1).So $C_1 = ([0,3],[0,2][0,1],[3,1])$. At time 2, since $u_{out}(1) = 3 \geq \tau_{out}(1) = 1$, also according to the maximum spike consumption strategy, neuron out fires by the rule $(a^2, a) \to a(1)$ and sends one spike to the environment through channel (1). The configuration of System Π at this time is $C_1 = ([0,3],[0,2][0,1],[1,1])$. At time 3, since $u_{out}(2) = 1 \geq \tau_{out}(2) = 1$, rule $(a, \lambda) \to \lambda$ is applied and removes one spike in feeding input unit and the system Π halts. Therefore, the spike train generated is "010".

3. Turing Universality of DTNP-MCIR Systems as Number-Generating/Accepting Device

In this section, we will explain the working mechanism of DTNP-MCIR systems as the number generating and number accepting mode. DTNP-MCIR systems proves its computational completeness by simulating register machines. More specifically, DTNP-MCIR systems can generate/accept all recursively enumerable sets of numbers (their family is called NRE).

A register machine can be defined as $M = (m, H, l_0, l_h I)$, where m is the number of registers, H indicates the set of instruction labels, l_0 corresponds to the start label, l_h corresponds to the halt label, I denotes the set of instructions. Each instruction in I corresponds to a label in H, the instructions in I have the following three forms:

(1) $l_i : (\mathrm{ADD}(r), l_j, l_k)$ (add 1 to register r and then move non-deterministically to one of the instructions with labels l_j, l_k).

(2) $l_j : (\mathrm{SUB}(r), l_j, l_k)$ (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise go to the instruction with label l_k).

(3) l_h : HALT (halting instruction).

3.1. DTNP-MCIR Systems as Number Generating Devices

Initially, every register in the register machine M is empty. The register machine can calculate the number n in the generation mode. It first starts with instruction l_0, and then applies a series of instructions until it reaches the halting instruction l_h. Finally, the number stored in the first register is the result calculated by the register machine. Usually the family NRE can be characterized by the register machine.

Theorem 1. *$N_2SNP_*^2 = NRE$.*

Proof. As we all know, $N_2SNP_*^2 \subseteq$ NRE. So we only need to prove NRE $\subseteq N_2SNP_*^2$ we consider a register machine $M = (m, H, l_0, l_h I)$ that works in generation mode, assuming that all registers except register 1 are empty, and register 1 never decrements during the calculation. □

We designed a DTNP-MCIR system Π_1 to simulate the register machine working in generating mode. The system Π_1 includes three modules: ADD module, SUB module, and FIN module. ADD module and SUB module used to simulate the ADD instruction and the SUB instruction respectively, and FIN module deals with the calculation results of the system Π_1.

We stipulate that each register r of register machine M corresponds to a neuron σ_r, note that there are no rules in neuron σ_r,numbers can be encoded in neuron σ_r. If the number stored in the register r is $n \geq 0$, then neuron σ_r contains 2n spikes. Each instruction l corresponds to a neuron σ_l, and some auxiliary neurons are introduced into models. Assume that there are no spikes in the feeding input unit of all auxiliary neurons, but neuron σ_{li} receives two spikes at the beginning. Each neuron has an

initial threshold for the initial threshold unit: (i) the initial threshold of each instruction neuron σ_l is a^2; (ii) each register neuron σ_r contains an initial threshold of a; (iii) the initial threshold of other neurons is uncertain. Because neuron σ_{li} gets two spikes, the system Π_1 begin to simulate the instruction n $l_i = \left(OP(r), l_j, l_k\right)$ (OP represents one of ADD or SUB operations). Starting from the activated neuron σ_{li}, the simulation deals with neuron σ_r according to OP, and then two spikes are introduced to one of the neurons σ_{lj} and σ_{lk}. The simulation will continue until the neuron σ_{lh} fires, indicating that the simulation is completed [30]. In the calculation process, the spikes are sent to the environment twice by neuron σ_{out} at times t_1 and t_2 respectively, the calculation result is defined as the interval of $t_2 - t_1$ that is also the number contained in register1.

In order to verify that the system Π_1 can indeed simulate the register machine M correctly, we will explain how the ADD or SUB module simulates the ADD or SUB instruction, and how the FIN module outputs the calculation results.

(1) **ADD module** (shown in Figure 3)—simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$

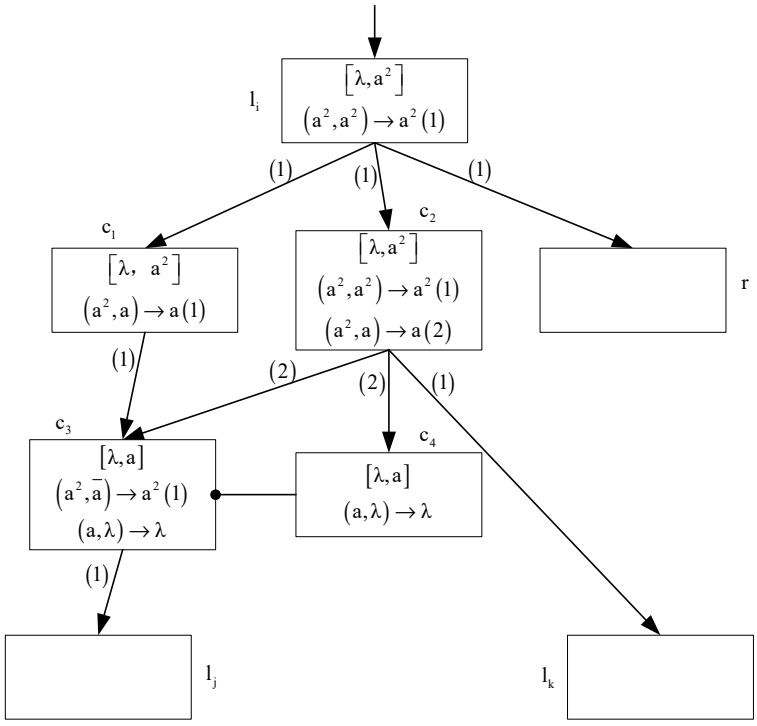


Figure 3. Module ADD simulating the ADD instruction $l_i : \left(ADD(r), l_j, l_k\right)$.

The system Π_1 starts from ADD instruction l_0. Suppose an ADD instruction $l_i : (ADD(r), l_j, l_k)$ is simulated at time t. At this time, neuron σ_{l_i} contains two spikes and the configuration of module ADD is $C_t = ([u_1, \tau_1], [u_2, \tau_2], \ldots, [u_7, \tau_7],)$, which involves 7 neurons $\sigma_{l_i}, \sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}, \sigma_{c_4}, \sigma_{l_j}, \sigma_{l_k}$ respectively. Thus, $C_t = ([2,2], [0,2], [0,2], [0,1], [0,1], [0,2], [0,2])$. Since $u_{li}(t) = 2 \geq \tau_{li}(t) = 2$, rule $\left(a^2, a^2\right) \to a^2(1)$ is applied in neuron σ_{l_i},then two spikes are sent to neurons $\sigma_{c_1}, \sigma_{c_2}, \sigma_r$ via channel (1). Because neuron σ_r receives two spikes, register r increases by one. Thus, $C_{t+1} = ([0,2], [2,2], [2,2], [0,1], [0,1], [0,2], [0,2])$.

At time t + 1, both feeding input unit and dynamic threshold unit of neuron σ_{c_1} and neuron σ_{c_2} contain two spikes, so they fire. Rule $\left(a^2, a\right) \to a(1)$ in neuron σ_{c_1} is applied and send one spike to neuron σ_{c_3}. For neuron σ_{c_2}, rules $\left(a^2, a^2\right) \to a^2(1)$ and $\left(a^2, a\right) \to a(2)$ both satisfy the firing condition, and both consume an equal number of spikes, so one of them will be selected non-deterministically. We consider the following two cases:

(i) At time t + 1, if rule $\left(a^2, a^2\right) \rightarrow a^2(1)$ is applied, neuron σ_{c_2} sends two spikes to neuron σ_{l_k} via channel (1). Then neuron σ_{l_k} receives two spikes, system Π_1 starts to simulate instruction l_k. Therefore $C_{t+2} = ([0,2],[0,2],[0,2],[1,1],[0,1],[0,2],[2,2])$. At time t + 3, since $u_{c_3}(t+2) = 1 \geq \tau_{c_3}(t+2) = 1$, the rule $(a, \lambda) \rightarrow \lambda$ of neuron σ_{c_3} is enabled and removes the only spike in the feeding input unit.

(ii) At time t + 1, if rule $\left(a^2, a\right) \rightarrow a(2)$ is applied, neuron σ_{c_2} sends one spike to neurons σ_{c_3} and σ_{c_4} via channel (2). Thus, $C_{t+2} = ([0,2],[0,2],[0,2],[2,1],[1,1],[0,2],[0,2])$. Note that neuron σ_{c_4} is the inhibitory neuron of neuron σ_{c_3}, since $u_{c_4}(t+2) = \tau_{c_3}(t+2)$, inhibitory rule $\left(a^2, \overline{a}\right) \rightarrow a^2(1)$ is enabled, neuro n σ_{c_3} sends two spikes to neuron σ_{l_j}. This means that the system Π_1 starts to simulate instruction l_j. The configuration of Π_1 at this time is $C_{t+3} = ([0,2],[0,2],[0,2],[0,1],[0,1],[2,2],[0,2])$.

Therefore, ADD instructions can be correctly simulated by the ADD module. Since neuron σ_{l_i} has two spikes, two spikes are added to neuron σ_r, and then neuron σ_{l_j} and σ_{l_k} are selected non-deterministically.

(2) **SUB module** (shown in Figure 4)—simulating a SUB instruction $l_j : (SUB(r), l_j, l_k)$.

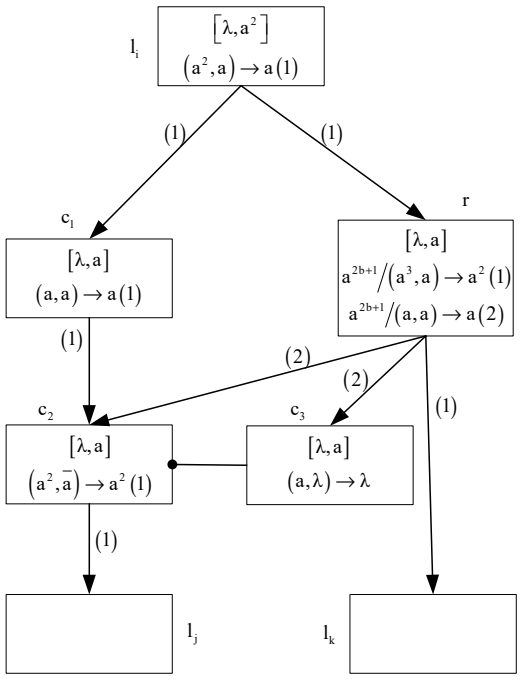


Figure 4. Module SUB simulating a SUB instruction $l_j : (SUB(r), l_j, l_k)$.

Suppose that the system Π_1 starts to simulate the SUB instruction $l_j : (SUB(r), l_j, l_k)$ at time t, and neuron σ_{l_i} contains two spikes at the same time. The configuration of module SUB is $C_t' = ([u_1, \tau_1], [u_2, \tau_2], \ldots, [u_7, \tau_7],)$, which involves 7 neurons $\sigma_{l_i}, \sigma_r, \sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}, \sigma_{l_j}, \sigma_{l_k}$ respectively. Thus, $C_t' = ([2,2],[2n,1],[0,1],[0,1],[0,1],[0,2],[0,2])$. Since $u_{l_i}(t) = 2 \geq \tau_{l_i}(t) = 2$, neuron σ_{l_i} fires and sends a spike to neuron σ_r and σ_{c_1} via channel (1). Therefore $C_{t+1}' = ([0,2],[2n+1,1],[1,1],[0,1],[0,1],[0,2],[0,2])$. Based on the number of spikes in neuron σ_r, there are two cases:

(i) If feeding input unit of neuron σ_r contains $2n \geq 0$ spikes at time t, then at time t + 1, neuron σ_r contains $2n + 1 \geq 3$ spikes. At time t + 2, rule $a^{2b+1}/\left(a^3, a\right) \rightarrow a^2(1)$ will be enabled and neuron σ_r will send two spikes to neurons σ_{l_k} via channel (1). This indicates that system Π_1 starts to simulate the instruction l_k of register machine M. In addition,

rule $(a, a) \rightarrow a(1)$ satisfies firing condition, neuron σ_{c_1} sends a spike to neuron σ_{c_2}. Thus $C_{t+2}' = ([0,2], [2n-2,1], [0,1], [1,1], [0,1], [0,2], [2,2])$. At time t+3, the forgetting rule $(a, \lambda) \rightarrow \lambda$ of neuron σ_{c_2} is enabled and removes the only spike in the feeding input unit. Therefore, $C_{t+3}' = ([0,2], [2n-2,1], [0,1], [0,1], [0,1], [0,2], [2,2])$.

(ii) If feeding input unit of neuron σ_r does not contain spikes at time t, then at time t + 1, neuron σ_r contains only one spike. At time t + 2, rule $a^{2b+1}/(a, a) \rightarrow a(1)$ will be applied and neuron σ_r will transmit one spike to neurons σ_{c_2} and σ_{c_3} via channel (2). At the same time neuron σ_{c_1} also transmits a spike to neuron σ_{c_2}. So $C_{t+2}' = ([0,2], [0,1], [0,1], [2,1], [1,1], [0,2], [0,2])$. Note that neuron σ_{c_3} is the inhibitory neuron of neuron σ_{c_2}. At time t + 3, the forgetting rule $(a, \lambda) \rightarrow \lambda$ of neuron σ_{c_3} is enabled and removes the only spike in the feeding input unit. Moreover, since $u_{c_3}(t+2) = \tau_{c_2}(t+2)$, rule $\left(a^2, \overline{a}\right) \rightarrow a^2(1)$ is applied, neuron σ_{c_2} sends two spikes to neuron σ_{l_j}. This means that the system Π_1 starts to simulate instruction l_j. Thus, $C_{t+2}' = ([0,2], [0,1], [0,1], [0,2], [0,1], [2,2], [0,2])$.

It can be seen from the above description that the SUB instruction can be correctly simulated by the SUB module. The system Π_1 starts with neuron σ_{l_i} contains two spikes, then determine whether neuron σ_{l_k} or σ_{l_j} receives two spikes according to whether neuron σ_r contains spikes.

(3) **Module FIN** (shown in Figure 5)—outputting the result of computation

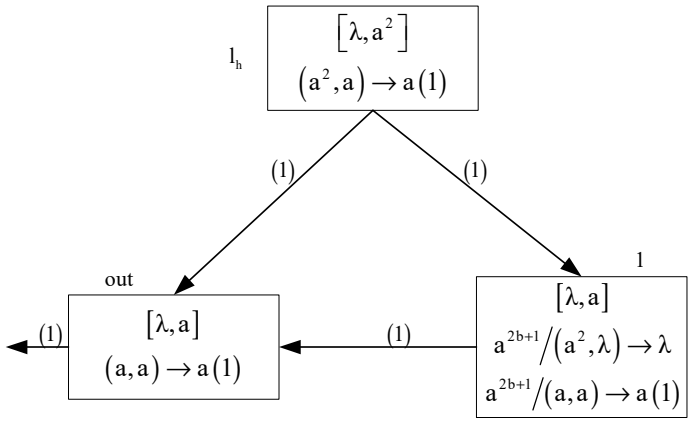


Figure 5. FIN module.

We assume that neuron σ_1 contains 2n spikes. Since neuron σ_{l_h} has two spikes at time t, it means register machine M halts, the rule $\left(a^2, a\right) \rightarrow a(1)$ is enabled and sends a spike to neurons σ_1 and σ_{out}. Therefore, the number of spikes in neuron σ_1 becomes an odd number. At time t + 1, rule $(a, a) \rightarrow a(1)$ in neuron σ_{out} is enabled and sends the first spike to the environment. Moreover, both rule $a^{2b+1}/\left(a^2, \lambda\right) \rightarrow \lambda$ and rule $a^{2b+1}/(a, a) \rightarrow a(1)$ can be applied in neuron σ_1. However, according to neuron's maximum spike consumption strategy, forgetting rule $a^{2b+1}/\left(a^2, \lambda\right) \rightarrow \lambda$ is applied first and two spikes are eliminated. This process will repeat until there is only a spike in neuron σ_1. At time $t + n$, neuron σ_1 contains only a spike. Thus, the rule $a^{2b+1}/(a, a) \rightarrow a(1)$ is applied and transmits a spike to neuron σ_{out}. At time $t + n + 1$, rule $(a, a) \rightarrow a(1)$ in neuron σ_{out} is enabled and sends the second spike to the environment. The time interval between two spikes being transmitted to the environment is $n = (t + 1) - (t + n + 1)$, which is the number in register σ_1 when the register machine halts.

According to the above description, it can be found that the system Π_1 can correctly simulate register machine M working in generating module. Therefore, Theorem 1 holds.

3.2. Turing Universality of Systems Working in the Accepting Mode

In the following, we will prove the universal of DTNP-MCIR systems as number accepting device.

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

Proof. A DTNP-MCIR system Π_2 is constructed to simulate a register machine $M = (m, H, l_0, l_h, I)$ working in the accepting mode. The setting of the initial threshold is the same as that used in the proof of Theorem 1 and initially any auxiliary neurons do not contain spikes. The proof is based on a modification to the proof of Theorem 1. System Π_2 contains a deterministic ADD module, a SUB module, and an INPUT module. □

Figure 6 shows the input module where neuron σ_{in} is used to read the spike train $10^{n-1}1$ from the environment. The number accepted by system Π_2 is the interval ($n = (n+1) - 1$) between two spikes in the spike train.

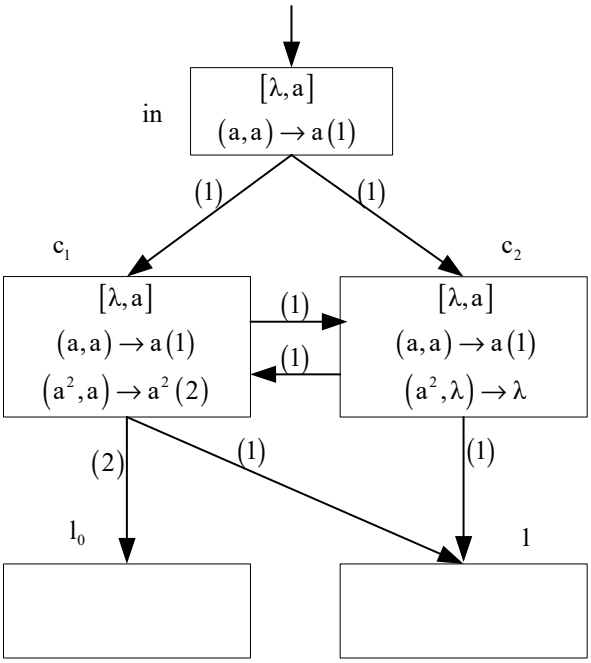


Figure 6. The INPUT module of Π_2.

Suppose that neuron σ_{in} imports the first spike from the environment at time t. Therefore, the initial configuration of System Π_2 is $C_t = ([1,1],[0,1],[0,1],[0,2],[0,1])$, which involves 5 neurons $\sigma_{l_i}, \sigma_{c_1}, \sigma_{c_2}, \sigma_{l_0}, \sigma_1$ respectively. Then rule $(a,a) \to a(1)$ in neuron σ_{in} is enabled and sends a spike to neurons σ_{c_1} and σ_{c_2} via channel (1). Thus $C_{t+1} = ([0,1],[1,1],[1,1],[0,2],[0,1])$. At time $t+2$, the rule $(a,a) \to a(1)$ in neurons σ_{c_1} and σ_{c_2} is applied. From time t + 2 to the second spike is imported, neurons σ_{c_1} and σ_{c_2} both send a spike to neuron σ_1 and exchange a spike with each other via channel (1). Thus neuron σ_1 receives a total of 2n spikes from time t + 2 to time t + n + 1, which also indicates that the number contained in register 1 is n. At time t + n + 1, neuron σ_{in} receives the second spike. The configuration of system Π_2 at this time is $C_{t+n+1} = ([1,1],[1,1],[1,1],[2n,2],[0,1])$. At the next moment, the rule $(a,a) \to a(1)$ is enabled again and neuron σ_{in} sends a spike to neurons σ_{c_1} and σ_{c_2}. Thus, $C_{t+n+2} = ([0,1],[2,1],[2,1],[2n,2],[0,1])$. Since neurons σ_{c_1} and σ_{c_2} each contain two spikes, the forgetting rule $(a^2, \lambda) \to \lambda$ in neuron σ_{c_2} is enabled, so the two spikes in feeding input unit are removed. The rule $(a^2, a) \to a^2(2)$ in neuron σ_{c_1} is also applied and two spikes are transmitted to neuron σ_{l_0} via channel (2). Since neuron σ_{l_0} contains two spikes, system Π_2 starts to simulate instruction l_0.

In the acceptance mode, we use the deterministic ADD module as shown in Figure 7 to simulate the instruction $l_i : (ADD(r), l_j)$ in the register machine M. When neuron σ_{l_i} receives two spikes, rule $\left(a^2, a\right) \rightarrow a^2(1)$ is applied and two spikes is sent to neurons σ_{l_j} and σ_r. This means that system Π_2 starts to simulate instruction l_j and register r increases by 1. The SUB module shown in Figure 4 is used to simulate the instruction $l_j : (SUB(r), l_j, l_k)$.

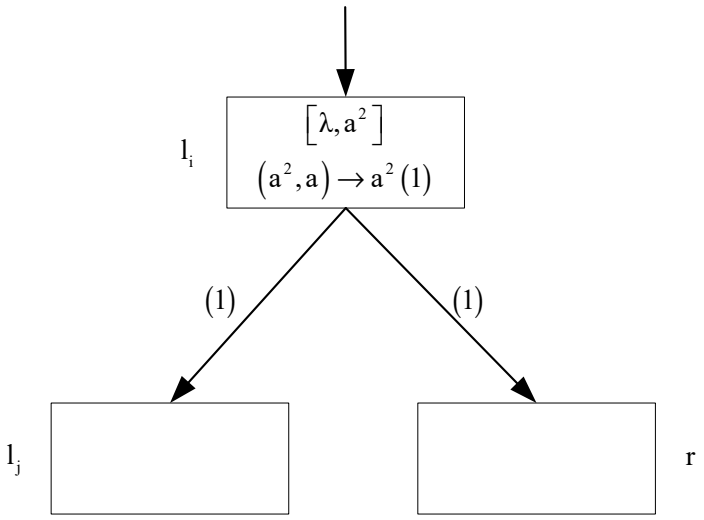


Figure 7. Deterministic ADD module, simulating $l_i : (ADD(r), l_j)$.

Module FIN is omitted, but the instruction σ_{l_h} remains in system Π_2. When neuron σ_{l_h} receives two spikes, it indicates that the halt instruction l_h is reached, that is the register machine M halts.

Through the above description, we prove that system Π_2 can correctly simulate the register machine M working in the accepting mode and can find that neurons of system Π_2 contain at most two rules, Therefore, Theorem 2 holds.

4. DTNP-MCIR Systems as Function Computing Devices

In this part, we will discuss the ability of a small universal DTNP-MCIR system to calculate functions. A register machine $M = (m, H, l_0, l_h, I)$ is used to compute the function $f : N^k \rightarrow N$. Initially we assume that all registers are empty, k arguments are introduced into k registers. In general, only the first two registers are used. Then the register machine starts from the instruction l_0 and continues to calculate until it reaches the halt instruction l_h. Finally, the function value is stored in a special register r_t. $(\varphi_0, \varphi_1, \ldots)$ is defined as a fixed admissible enumeration of the unary partial recursive functions. We think that a register machine is universal when there is a recursive function g that makes $\varphi_X(y) = M_u(g(x), y)$ for all natural numbers x, y.

Korec [31] proposed a small universal register machines $M_u = (8, H, l_0, l_h, I)$. M_u contains 8 registers (labeled from 0 to 8) and 23 instructions. By introducing two numbers g(x) and y into registers 1 and 2, respectively, $\varphi_x(y)$ can be computed by M_u. When M_u stops, the function value is stored in register 0. In order to simplify the register machine M_u, we introduce a new register 8 and replace the halt instruction with three new instructions: $l_{22} : (SUB(0), l_{23}, l_h)$; $l_{23} : (ADD(8), l_{22})$; $l_h : HALT$. We define the modified register machine as M_u', as shown in Table 1. We will design a small universal DTNP-MCIR system to simulate the register machine M_u'.

Table 1. Universal register machines M_u'.

$l_0 : (SUB(1), l_1, l_2)$	$l_1 : (ADD(7), l_0)$
$l_2 : (ADD(6), l_3)$	$l_3 : (SUB(5), l_2, l_4)$
$l_4 : (SUB(6), l_5, l_3)$	$l_5 : (ADD(5), l_6)$
$l_6 : (SUB(7), l_7, l_8)$	$l_7 : (ADD(1), l_4)$
$l_8 : (SUB(6), l_9, l_0)$	$l_9 : (ADD(6), l_{10})$
$l_{10} : (SUB(4), l_0, l_{11})$	$l_{11} : (SUB(5), l_{12}, l_{13})$
$l_{12} : (SUB(5), l_{14}, l_{15})$	$l_{13} : (SUB(2), l_{18}, l_{19})$
$l_{14} : (SUB(5), l_{16}, l_{17})$	$l_{15} : (SUB(3), l_{18}, l_{20})$
$l_{16} : (ADD(4), l_{11})$	$l_{17} : (ADD(2), l_{21})$
$l_{18} : (SUB(4), l_0, l_{22})$	$l_{19} : (SUB(0), l_0, l_{18})$
$l_{20} : (ADD(0), l_0)$	$l_{21} : (ADD(3), l_{18})$
$l_{22} : (SUB(0), l_{23}, l_h)$	$l_{23} : (ADD(8), l_{22})$
l_h : HALT	

Theorem 3. *There is a small universal DTNP-MCIR system having 73 neurons for computing functions.*

Proof. We design a DTNP-MCIR system Π_3 to simulate the register machine M_u'. System Π_3 contains an INPUT module, an OUTPUT module, and some ADD and SUB modules to simulate the ADD and SUB instructions of M_u', respectively. The INPUT module is used to read the spike train from the environment, and the OUTPUT module deal with calculation result which is placed in register 8. Each register r and each instruction l_i of M_u' corresponds to a neuron σ_r and σ_{l_i} respectively. If the feeding input unit of neuron σ_r contains 2n spikes, it means that the register r contains the number n. When neuron σ_{l_i} receives two spikes, the simulation of instruction l_i is started. Initially, all neurons in system Π_3 are assumed to be empty. □

The INPUT module is shown in Figure 8. The spike train $10^{g(x)}10^{y}1$ is read from the environment by it, and 2g(x) spikes and 2y spikes will be stored in neurons σ_1 and σ_2 respectively. We do not use inhibitory rules in this module, but make full use of the function of multiple channels. Multiple channels have played a major role in saving neurons and improving system operation efficiency.

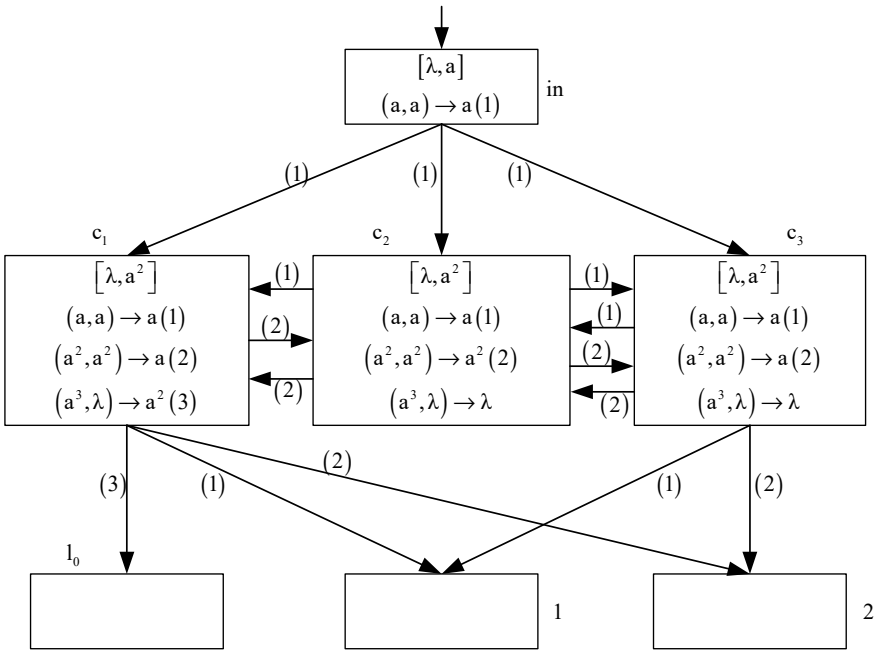


Figure 8. Module INPUT.

Suppose that neuron σ_{in} receives the first spike form the environment at time t_1.At time t_1+1, rule $(a,a) \to a(1)$ is applied, and neuron σ_{in} sends a spike to neurons σ_{c_1},σ_{c_2} and σ_{c_3}. Since neurons σ_{c_1}, σ_{c_2} and σ_{c_3} all contain a spike, the rule $(a,a) \to a(1)$ in three neurons is enabled at time t_1+2. That is, neuron σ_{c_2} sends a spike to neuron σ_{c_1}, neuron σ_{c_2} and neuron σ_{c_3} exchange a spike with each other, as well as neuron σ_{c_1} and neuron σ_{c_2} each send a spike to neuron σ_1 via channel (1). This process will repeat until the second spike reaches neuron σ_{in}. Therefore, neuron σ_1 receives two spikes in each moment from time t_1+2 to time $t_1+g(x)+1$ and imports a total of $2g(x)$ spikes. (That is, the number stored in neuron σ_1 is $g(x)$).

Assuming that the time for the second spike to reach neuron σ_{in} is t_2, in fact $t_2 = t_1+g(x)+1$. Similarly, rule $(a,a) \to a(1)$ in neuron σ_{in} is applied to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3} at time t_2+1. Then neurons σ_{c_1}, σ_{c_2} and σ_{c_3} each contain two spikes, so rule $\left(a^2,a^2\right) \to a(1)$ in neurons σ_{c_1} and σ_{c_3} is enabled and rule $\left(a^2,a^2\right) \to a^2(1)$ is applied in neurons σ_{c_2} at time t_2+2. Then neurons σ_{c_1} and σ_{c_3} respectively send a spike to neuron σ_2 via channel(2) and exchange spikes with neuron σ_{c_2} each step from this moment.

Neuron σ_2 receives a total of 2y spikes from time t_2+2 to time t_2+y+1 (That is, the number stored in neuron σ_2 is y). When the third spike is imported by neuron σ_{in}, the next moment neurons σ_{c_1},σ_{c_2}, and σ_{c_3} will each contain three spikes. Three spikes are consumed by forgetting rule $\left(a^3,\lambda\right) \to \lambda$ in neuron σ_{c_2} and σ_{c_3}. Rule $\left(a^3,\lambda\right) \to a^2(3)$ in neuron σ_{c_1} is applied to transmit two spikes to neuron σ_{l_0} via channel (3), this indicates that the system Π_3 starts to simulate the initial instruction l_0.

By observing Table 1, we can find that all ADD instructions have the form $l_i : (ADD(r), l_j)$, therefore, we use the deterministic ADD module shown in Figure 7 to simulate the ADD instructions. Its working mechanism has been clarified in the proof of Theorem 2.

In addition, the SUB instruction $l_j : (SUB(r), l_j, l_k)$ can be simulated by the SUB module in Figure 4. The working mechanism of the SUB module has been discussed during the proof of Theorem 1.

When neuron σ_{l_h} receives two spikes at time t, the calculation of the system Π_3 halts. The OUTPUT module shown in Figure 9 is used to deal with the calculation results. We assume that neuron σ_8 contains 2n spikes. The configuration of OUTPUT module at time t is $C_t''' = ([u_1,\tau_1],[u_2,\tau_2],\ldots,[u_5,\tau_5],)$, which involves fives neurons σ_{l_h}, σ_{c_1}, σ_{c_2}, σ_8, σ_{out} respectively. Thus, $C_t''' = ([2,2],[0,1],[0,1],[2n,1],[0,1])$. Rule $\left(a^2,a\right) \to a(1)$ is enabled, and neuron σ_{l_h} sends one spike to neurons σ_{c_1}, σ_{c_2} and σ_8 via channel (1) at time t. Therefore, $C'''_{t+1} = ([0,2],[1,1],[1,1],[2n+1,1],[0,1])$. Since neuron σ_{c_2} is the inhibitory neuron of neuron σ_{c_1} and $u_{c_2}(t+1) = \tau_{c_1}(t+1)$, rule $(a,\overline{a}) \to a(1)$ is applied, neuron σ_{c_1} sends a spike to neuron σ_{out}. Rule $(a,a) \to a(1)$ in neuron σ_{c_2} and rule $a^{2b+1}/\left(a^3,a\right) \to a(1)$ in neuron σ_8 are also enabled. Although the rule $a^{2b+1}/\left(a^3,a\right) \to a(1)$ consumes three spikes of neuron σ_8, because neuron σ_8 and neuron σ_{c_2} exchange one spike with each other via channel (1), neuron σ_8 only consumes two spikes each moment. Thus, $C'''_{t+2} = ([0,2],[0,1],[1,1],[2n-1,1],[1,1])$. Rule $(a,a) \to a(1)$ is applied in neuron σ_{out}, then the first spike is sent to the environment at time t + 2. Rule $a^{2b+1}/\left(a^3,a\right) \to a(1)$ will be executed continuously until neuron σ_8 contains only one spikes. At time t + n + 1, neuron σ_8 contains only a spike, rule $a^{2b+1}/(a,a) \to a(2)$ is applied and neuron σ_8 transmits a spike to neuron σ_{out} via channel (2). At time t + n + 2, rule $(a,a) \to a(1)$ is applied, neuron σ_{out} sends the second spike to the environment. According to the definition in Section 2, the result of the system Π_3 is $n = (t+n+2)-(t+2)$, which is also the number contained in register 8.

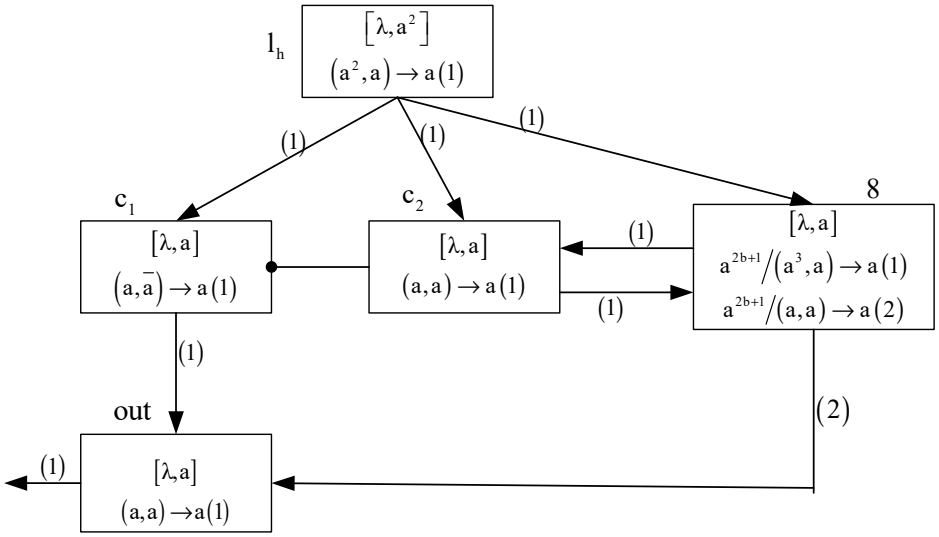


Figure 9. OUTPUT module.

From the above description, it can be seen that the system Π_3 can correctly simulate the register machine M_u' calculation function. System Π_3 contains a total of 81 neurons: (1) the INPUT module has 3 auxiliary neurons; (2) the OUTPUT module has 2 auxiliary neurons; (3) each SUB module has 3 auxiliary neurons, for a total of 42 neurons; (4) 9 neurons correspond to 9 registers; (5) 25 neurons correspond to 25 instructions.

We can further reduce the number of neurons in system Π_3 by combining instructions. Here are three cases:

Case 1. For the sequence of two consecutive ADD instructions $l_{17} : (\mathrm{ADD}(2), l_{21})$ and $l_{21} : (\mathrm{ADD}(3), l_{18})$, we can use the model ADD-ADD in Figure 10 to simulate so that neuron $\sigma_{l_{21}}$ corresponding to instruction l_{21} can be saved.

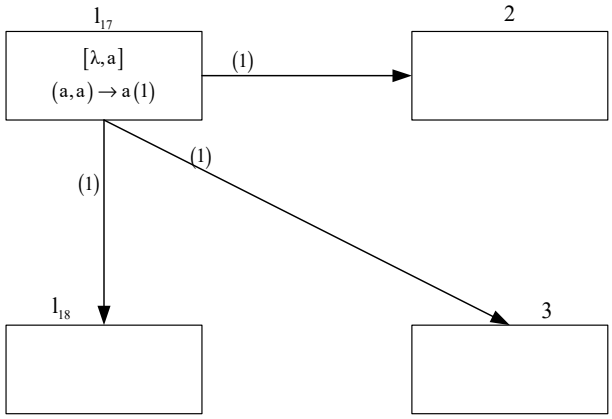


Figure 10. The model simulating consecutive ADD-ADD instructions $l_{17} : (\mathrm{ADD}(2), l_{21})$ and $l_{21} : (\mathrm{ADD}(3), l_{18})$.

Case 2. We can also combine instructions $l_{15} : (\mathrm{SUB}(3), l_{18}, l_{20})$ and $l_{20} : (\mathrm{ADD}(0), l_0)$ to form one instruction $l_{15} : (\mathrm{SUB}(3), l_{18}, (\mathrm{ADD}(0), l_0))$ so that $\sigma_{l_{20}}$ will be saved. The SUB-ADD-1 model shown in Figure 11 can be used to simulate newly formed instructions.

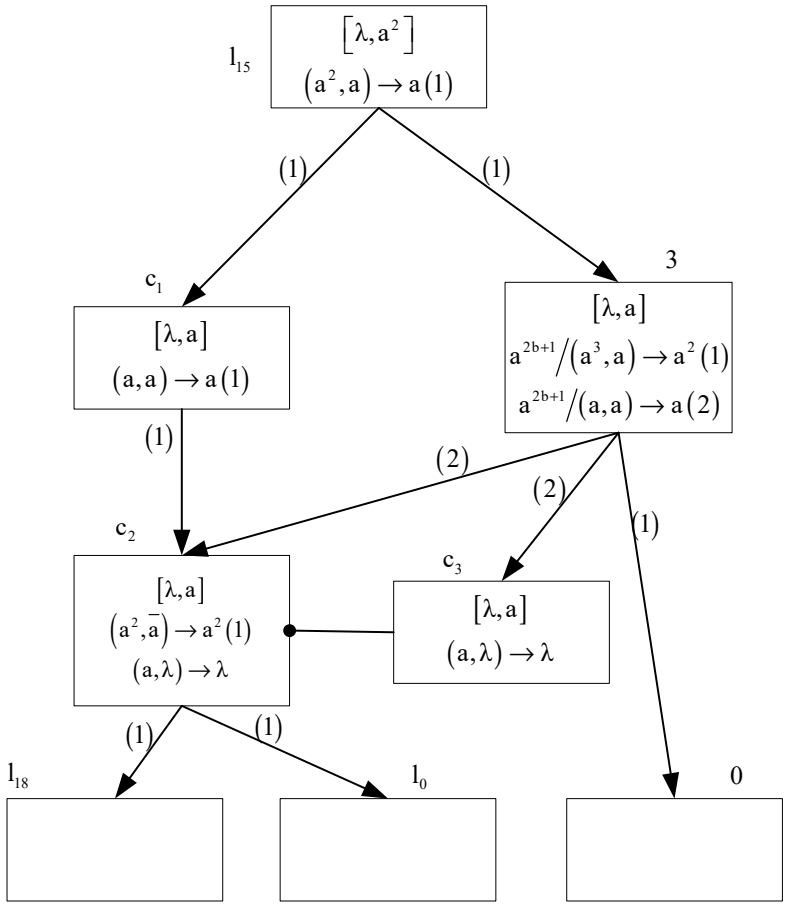


Figure 11. The model simulating consecutive SUB-ADD-1 instructions $l_{15} : (\mathrm{SUB}(3), l_{18}, l_{20})$ and $l_{20} : (\mathrm{ADD}(0), l_0)$.

Case 3. The following six pairs of ADD and SUB instructions can be combined.

$$l_0 : (\mathrm{SUB}(1), l_1, l_2), l_1 : (\mathrm{ADD}(7), l_0);$$
$$l_4 : (\mathrm{SUB}(6), l_5, l_3), l_5 : (\mathrm{ADD}(5), l_6);$$
$$l_6 : (\mathrm{SUB}(7), l_7, l_8), l_7 : (\mathrm{ADD}(1), l_4);$$
$$l_8 : (\mathrm{SUB}(6), l_9, l_0), l_9 : (\mathrm{ADD}(6), l_{10});$$
$$l_{14} : (\mathrm{SUB}(5), l_{16}, l_{17}), l_{16} : (\mathrm{ADD}(4), l_{11});$$
$$l_{22} : (\mathrm{SUB}(0), l_{23}, l_h), l_{23} : (\mathrm{ADD}(8), l_{22});$$

By observing the above six pairs of instructions, it can be seen that the latter ADD instruction is at the position of the first exit of the previous SUB instruction. Therefore, the combination instructions can be expressed as $l_i : \left(\mathrm{SUB}(r_1), l_j, l_k\right), l_j : \left(\mathrm{ADD}(r_2), l_g\right)$. In this way, we can save six neurons. The recombined SUB and ADD instructions can be simulated by the SUB-ADD-2module in Figure 12.

In summary, we can save eight neurons in total by combining some ADD/SUB instructions. The recombined instructions can be simulated with ADD-ADD, SUB-ADD-1and SUB-ADD-2modules respectively. Therefore, the number of neurons in system Π_3 dropped from 81 to 73. So far, we have completed the proof of Theorem 3.

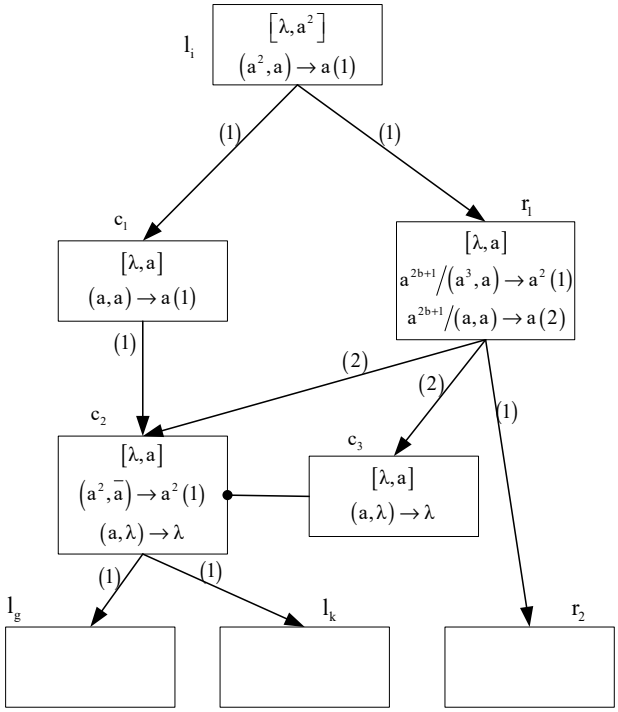


Figure 12. The model simulating consecutive SUB-ADD-2 instructions $l_i : (\mathrm{SUB}(r_1), l_j, l_k), l_j : (\mathrm{ADD}(r_2), l_g)$.

In order to further illustrate the computing power of DTNP-MCIR systems, we compared it with some computing models in term of small number of computing units. From Table 2, we can see that DTNP systems, SNP systems, SNP-IR systems and recurrent neural networks need 109,67,100 and 886 neurons respectively to achieve Turing universality for computing function. However, DTNP-MCIR systems need fewer neurons than them. Although SNP-MC systems require only 38 neurons to compute function, which is much less than the computing unit required by DTNP-MCIR systems, they have different working modes. SNP-MC systems work in asynchronous mode but DTNP-MCIR systems work in synchronous mode. Therefore, the comparison indicates that DTNP-MCIR systems need fewer neurons than most of modes. In addition, Table 3 gives the full name of the comparative model abbreviation.

Table 2. The comparison of different computing models in the term of small number of computing units.

Computing Models	Computing Functions
DTNP-MCIR systems	73
DTNP systems [29]	109
SNP systems [32]	67
SNP-IR systems [28]	100
Recurrent neural networks [33]	886
SNP-MC systems [34]	38

Table 3. Abbreviations and corresponding full names of some systems cited in the paper.

DTNP systems [29]	Dynamic threshold neural P systems
SNP-IR systems [28]	Spiking neural P systems with inhibitory rules
SNP-MC systems [27]	Spiking neural P systems with multiple channels
SNP systems [32]	Smaller universal spiking neural P systems
SNP-MC systems [34]	Small universal asynchronous spiking neural P systems with multiple channels.

5. Conclusions and Further Work

Inspired by SNP systems with inhibitory rules (SNP-IR systems) and the learning of SNP systems with multiple channels (SNP-MC systems), this paper proposes a dynamic threshold neural P system with inhibitory rules and multiple channels (DTNP-MCIR systems). In fact, dynamic threshold neural P systems (DTNP systems) had been researched and proved to be Turing universal number generating/accepting device. Our original intention to construct DTNP-MCIR systems is to more fully simulate the actual situation of neurons communicating through synapses, and also to show the use of inhibitory rules and multiple channels in DTNP systems. In addition, we have optimized DTNP systems. The form of firing rules is $E_i / (a^u, a^\tau) \rightarrow a^p$, where $u \geq 1, \tau \geq 0, p \geq 0$, and $p \leq u$. However, τ and p are always equal in the DTNP system, we think it lacks generality and improve it; For the rule $(a^u, a^\tau) \rightarrow a^p$ of neuron σ_r in the SUB module, If neuron σ_r contains 2n spikes at time t and $2n \geq u$, then neuron σ_r can be fired at time t, which is unreasonable for the entire system. Therefore, we improve the form of firing rules as $a^{2b+1} / (a^u, a^\tau) \rightarrow a^p(1)$, where a^{2b+1} is a regular expression we introduced, which means that neuron σ_r can be fired only when the number of spikes is an odd number.

In the future, we want to research whether DTNP-MCIR systems can be combined with certain algorithms, we think that the two data units of DTNP-MCIR systems can be used as two parameter inputs, the use of inhibitory rules and multiple channels make them have stronger control capabilities. Moreover, because DTNP-MCIR systems are a distributed parallel computing model, they can greatly improve the computational efficiency of algorithms. Future work will focus on using DTNP-MCIR systems to solve real-world problems, such as image processing and data clustering

Author Contributions: Conceptualization, X.Y. and X.L.; methodology, X.Y.; formal analysis, X.Y.; writing—original draft preparation, X.Y.; writing—review and editing, X.Y.; visualization, X.L.; funding acquisition, X.Y. and X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research project is supported by National Natural Science Foundation of China (61876101, 61802234, 61806114), Social Science Fund Project of Shandong Province, China (16BGLJ06, 11CGLJ22), Natural Science Fund Project of Shandong Province, China (ZR2019QF007), Postdoctoral Project, China (2017M612339, 2018M642695), Humanities and Social Sciences Youth Fund of the Ministry of Education, China (19YJCZH244), Postdoctoral Special Funding Project, China (2019T120607).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Păun, G. Computing with membranes. *J. Comput. Syst. Sci.* **2000**, *61*, 108–143. [CrossRef]
2. Song, T.; Gong, F.; Liu, X. Spiking neural P systems with white hole neurons. *IEEE Trans. NanoBiosci.* **2016**, *15*, 666–673. [CrossRef]
3. Freund, R.; Păun, G.; Pérez-Jiménez, M.J. Tissue-like P systems with channel-states. *Theor. Comput. Sci.* **2005**, *330*, 101–116. [CrossRef]
4. Song, T.; Pan, L.; Wu, T. Spiking neural P Systems with leaning functions. *IEEE Trans. NanoBiosci.* **2019**, *18*, 176–190. [CrossRef] [PubMed]
5. Cabarle, F.G.C.; Adorna, H.N.; Jiang, M.; Zeng, X. Spiking neural P systems with scheduled synapses. *IEEE Trans. NanoBiosci.* **2017**, *16*, 792–801. [CrossRef] [PubMed]
6. Song, T.; Pan, L. Spiking neural P systems with rules on synapses working in maximum spiking strategy. *IEEE Trans. NanoBiosci.* **2015**, *14*, 465–477. [CrossRef]
7. Ionescu, M.; Păun, G.; Yokomori, T. Spiking neural P systems. *Fund. Inform.* **2006**, *71*, 279–308.
8. Song, T.; Pan, L. Spiking neural P systems with request rules. *Neurocomputing* **2016**, *193*, 193–200. [CrossRef]
9. Zeng, X.; Zhang, X. Spiking Neural P Systems with Thresholds. *Neural Comput.* **2014**, *26*, 1340–1361. [CrossRef]
10. Zhao, Y.; Liu, X. Spiking Neural P Systems with Neuron Division and Dissolution. *PLoS ONE* **2016**, *11*, e0162882. [CrossRef]
11. Wang, J.; Shi, P.; Peng, H.; Perez-Jimenez, M.J.; Wang, T. Weighted Fuzzy Spiking Neural P Systems. *IEEE Trans. Fuzzy Syst.* **2012**, *21*, 209–220. [CrossRef]

12. Jiang, K.; Song, T.; Pan, L. Universality of sequential spiking neural P systems based on minimum spike number. *Theor. Comput. Sci.* **2013**, *499*, 88–97. [CrossRef]
13. Zhang, X.; Luo, B. Sequential spiking neural P systems with exhaustive use of rules. *Biosystems* **2012**, *108*, 52–62. [CrossRef] [PubMed]
14. Cavaliere, M.; Ibarra, O.H.; Păun, G.; Egecioglu, O.; Ionescu, M.; Woodworth, S. Asynchronous spiking neural P systems. *Theor. Comput. Sci.* **2009**, *410*, 2352–2364. [CrossRef]
15. Song, T.; Pan, L.; Păun, G. Asynchronous spiking neural P systems with local synchronization. *Inf. Sci.* **2013**, *219*, 197–207. [CrossRef]
16. Song, T.; Zou, Q.; Liu, X.; Zeng, X. Asynchronous spiking neural P systems with rules on synapses. *Neurocomputing* **2015**, *151*, 1439–1445. [CrossRef]
17. Păun, G. Spiking neural P systems with astrocyte-like control. *J. Univers. Comput. Sci.* **2007**, *13*, 1707–1721.
18. Wu, T.; Păun, A.; Zhang, Z.; Pan, L. Spiking Neural P Systems with Polarizations. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *29*, 3349–3360.
19. Păun, A.; Păun, G. Small universal spiking neural P systems. *BioSystems* **2007**, *90*, 48–60. [CrossRef]
20. Cabarle, F.; Adorna, H.; Perez-Jimenez, M.; Song, T. Spiking neuron P systems with structural plasticity. *Neural Comput. Appl.* **2015**, *26*, 1905–1917. [CrossRef]
21. Peng, H.; Chen, R.; Wang, J.; Song, X.; Wang, T.; Yang, F.; Sun, Z. Competitive spiking neural P systems with rules on synapses. *IEEE Trans. NanoBiosci.* **2017**, *16*, 888–895. [CrossRef] [PubMed]
22. Xiong, G.; Shi, D.; Zhu, L.; Duan, X. A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems. *Math. Probl. Eng.* **2013**, *2013*, 1–13. [CrossRef]
23. Wang, T.; Zhang, G.X.; Zhao, J.B.; He, Z.Y.; Wang, J.; Pérez-Jiménez, M.J. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Trans. Power Syst.* **2015**, *30*, 1182–1194. [CrossRef]
24. Peng, H.; Wang, J.; Ming, J.; Shi, P.; Pérez-Jiménez, M.J.; Yu, W.; Tao, C. Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems. *IEEE Trans. Smart Grid.* **2018**, *9*, 4777–4784. [CrossRef]
25. Peng, H.; Wang, J.; Shi, P.; Pérez-Jiménez, M.J.; Riscos-Núñez, A. An extended membrane system with active membrane to solve automatic fuzzy clustering problems. *Int. J. Neural Syst.* **2016**, *26*, 1–17. [CrossRef] [PubMed]
26. Zhang, G.; Rong, H.; Neri, F.; Pérez-Jiménez, M. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *Int. J. Neural Syst.* **2014**, *24*, 1440006. [CrossRef] [PubMed]
27. Peng, H.; Yang, J.; Wang, J.; Wang, T.; Sun, Z.; Song, X.; Lou, X.; Huang, X. Spiking neural P systems with multiple channels. *Neural Netw.* **2017**, *95*, 66–71. [CrossRef] [PubMed]
28. Peng, H.; Li, B.; Wang, J. Spiking neural P systems with inhibitory rules. *Knowl.-Based Syst.* **2020**, *188*, 105064. [CrossRef]
29. Peng, H.; Wang, J. Dynamic threshold neural P systems. *Knowl.-Based Syst.* **2019**, *163*, 875–884. [CrossRef]
30. Peng, H.; Wang, J. Coupled Neural P Systems. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 1672–1682. [CrossRef]
31. Korec, I. Small universal register machines. *Theor. Comput. Sci.* **1996**, *168*, 267–301. [CrossRef]
32. Zhang, X.; Zeng, X.; Pan, L. Smaller universal spiking neural P systems. *Fundam. Inf.* **2007**, *87*, 117–136.
33. Siegelmann, H.T.; Sontag, E.D. On the computational power of neural nets. *J. Comput. Syst.* **1995**, *50*, 132–150. [CrossRef]
34. Song, X.; Peng, H. Small universal asynchronous spiking neural P systems with multiple channels. *Neurocomputing* **2020**, *378*, 1–8. [CrossRef]