Hindawi
Complexity
Volume 2020, Article ID 8742308, 12 pages
https://doi.org/10.1155/2020/8742308

Research Article

Spiking Neural P Systems with Polarizations and Rules on Synapses

Suxia Jiang,[1] **Jihui Fan**,[1] **Yijun Liu**,[1] **Yanfeng Wang**,[1] **and Fei Xu**[2]

[1]*Henan Key Lab of Information Based Electrical Appliances, School of Electrical and Information Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, Henan, China*
[2]*Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China*

Correspondence should be addressed to Suxia Jiang; jiangsx913@126.com and Fei Xu; fei_xu@hust.edu.cn

Received 23 December 2019; Revised 27 May 2020; Accepted 16 June 2020; Published 9 July 2020

Academic Editor: Dimitri Volchenkov

Spiking neural P systems are a class of computation models inspired by the biological neural systems, where spikes and spiking rules are in neurons. In this work, we propose a variant of spiking neural P systems, called spiking neural P systems with polarizations and rules on synapses (PSNRS P systems), where spiking rules are placed on synapses and neurons are associated with polarizations used to control the application of such spiking rules. The computation power of PSNRS P systems is investigated. It is proven that PSNRS P systems are Turing universal, both as number generating and accepting devices. Furthermore, a universal PSNRS P system with 151 neurons for computing any Turing computable functions is given. Compared with the case of SN P systems with polarizations but without spiking rules in neurons, less number of neurons are used to construct a universal PSNRS P system.

1. Introduction

Membrane computing is a burgeoning branch of natural computing that develops new computation models based on the structure and functioning of living cells [1, 2]. Membrane systems (also called P systems) are distributed parallel computation models in membrane computing. There are three main classes of P systems, based on the structure of membranes inside living cells: cell-like P systems [3], tissue-like P systems [4], and neural-like P systems [5]. In the field of mathematical and theoretical computer science, P systems are used to investigate numerous types of problems, such as the Turing universality of the system [6, 7], complexity classes [8], numerical problems [9, 10], NP-complete problems [11–13], and P systems simulation [14, 15]. The reader can consult [16] for more comprehensive information about membrane computing. Up-to-date research results and open problems can be found on the membrane computing website http://ppage.psystems.eu.

Spiking neural P systems (SN P systems) are a class of neural-like P systems, inspired by the biological phenomenon of neurons conveying information by communicating with each other via identical electric impulses (spikes) [17]. In this type of systems, only one type of spike exists, and the information is encoded by the timing and number of spikes. SN P systems are a class of computation models that use spiking/forgetting rules, which is applied by matching the number of spikes with the regular expression. Many variants of SN P systems have been proposed based on various biological characteristics, such as scheduled synapses [18], structural plasticity [19, 20], thresholds [21, 22], and multiple channels [23]. Meanwhile, there are also extensive studies in the view of the mechanism of information communication between neurons, such as white hole neurons [24], request rules [25], inhibitory rules [26], and communication on request [27]. Most of these systems have been proven to have the equivalent computation power with Turing machine.

In addition, SN P systems have been widely investigated in the field of computer science, which can be used to produce binary and string languages [28–31] and simulate the registration machine [32, 33]. With the development of

the research, the SN P systems as small universal computing devices are studied [34, 35]. Moreover, SN P systems and their variants have been successfully implemented in real-life applications, for instance, logic gate design [36], image processing [37], fault diagnosis of electric power systems [38–40], optimization algorithm design for combinatorial optimization problems [41, 42], and robot control [43–46].

Recently, a variant of SN P systems called SN P systems with polarizations (PSN P systems) was proposed in [47], where the rules are controlled by three electrical charges $(-, 0, +)$ associated with each neuron, not by the regular expression; it is closer to biological reality. In PSN P systems, the use of rules is more limited compared with SN P systems, because only three electrical charges can be selected, but PSN P systems as number devices proved to be Turing universal. As universal computing devices, the computing process of PSN P systems is complicated, and a total of 164 neurons (computational resources) are consumed. It is an open problem whether to find such systems that consume less computational resources, such as using extended spiking rules and delay functions and so on. Herein, we are inspired by the functioning of reflex arcs; the brain sends different signals to different cells to elicit precise actions by the body when a stimulus enters the brain. This biological phenomenon was introduced into SN P systems in [48]; the rules are placed on the synapses, that is, the rules on synapses. However, the rules on the synapse are still triggered using the regular expression. If the use conditions of rules on synapses are weakened, what about the computation power of SN P systems with rules on synapses? Hence, it is also interesting to construct a variant of SN P systems with rules on synapses, where the rules are not controlled by the regular expression.

In this work, inspired by the open problems raised in [47], we put the rules on synapses, proposing SN P systems with polarizations and rules on synapses (PSNRS P systems, for short). In PSNRS P systems, the rules associated with each neuron are placed on synapses, and when the polarity of a neuron meets some rules of its synapses, the rules are activated, and the neuron sends different number of spikes and the same electrical charge to its neighboring neurons. Compared with neurons in PSN P systems that can only send the same number of spikes at some point to its neighboring neurons, the rules on synapses in PSNRS P systems will be more flexible. We investigate the computation power of PSNRS P systems working in the maximally parallel mode. The main contributions of the present work are summarized as follows:

(i) Rules on synapses and polarizations are considered in SN P systems; we construct a variant of SN P systems, called SN P systems with polarizations and rules on synapses (PSNRS P systems). In PSNRS P systems, polarization is used to control the application of spiking/forgetting rules associated with each synapse of the neurons.

(ii) The computation power of PSNRS P systems is investigated. Specifically, we prove that PSNRS P systems as accepting devices and generating devices achieve universality. Furthermore, a universal PSNRS P system with 151 neurons for computable functions is presented. Compared to the PSN P system, 13 neurons were reduced in computational resources.

The rest of this paper is organized as follows. The formal definition of PSNRS P systems is introduced in the next section. The computation power of PSNRS P systems as number generators and acceptors is investigated in Section 3. In Section 4, a small universal PSNRS P system for computable functions is given. Finally, conclusions and suggestions for further work are presented in Section 5.

2. Spiking Neural P Systems with Polarizations and Rules on Synapses

In this section, we first review some prerequisites. For details on the basic elements of membrane computing and automata theory, the reader can refer to [16, 49]. V^* is the set of all words over the alphabet, including the empty string λ. $V^* - \{\lambda\}$ corresponds to the set of nonempty words over V and is denoted by V^+. The family of sets of natural numbers computed by Turing machines is denoted by NRE.

In the following definition, the notions of polarity and rules on synapses will be used. Only a brief introduction is provided here; please refer to [47, 48, 50] for details.

A PSNRS P system of degree $m \geq 1$ is a construction as follows:

$$\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m, \text{syn}, \text{in}, \text{out}), \quad (1)$$

where

(i) $O = \{a\}$ is an alphabet and a denotes the spike

(ii) $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (\alpha_i, n_i)$, $1 \leq i \leq m$ (m is the number of neurons), where

  (a) $\alpha_i \in \{-, 0, +\}$ is the initial polarity of neuron σ_i
  (b) n_i is the number of spikes initially located in σ_i

(iii) $\text{syn} \in \{1, 2, 3, \ldots, m\} \times \{1, 2, 3, \ldots, m\}$; syn is the sets of synapses; each element is a pair of the form $((i, j), R_{(i,j)})$, where (i, j) indicates that there is a synapse connecting neurons σ_i and σ_j, with $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$, and $R_{(i,j)}$ is a finite set of rules of the following two forms:

  (a) $\alpha/a^c \longrightarrow a; \beta$, for $\alpha, \beta \in \{-, 0, +\}$, $c \geq 1$ (spiking rules)
  (b) $\alpha/a^c \longrightarrow \lambda; \beta$, for $\alpha, \beta \in \{-, 0, +\}$, $c \geq 1$ (forgetting rules)

(iv) $\text{in}, \text{out} \in \{1, 2, \ldots, m\}$ indicates the input and output neurons, respectively

The spiking rules are applied as follows. If neuron σ_i contains k spikes and has α charge ($a^k \in L(E)$, $k \geq c$), then the spiking rule $\alpha/a^c \longrightarrow a; \beta$ is enabled. Meanwhile, c spikes from neuron σ_i are consumed, and a spike and β charge are sent to neuron σ_j via a synapse (i, j). Note that

each synapse can have rules, and they do not affect each other, indicating that neuron σ_i sends a difference charge to adjacent neurons.

A forgetting rule $\alpha/a^c \longrightarrow \lambda;\beta$ is used when neuron σ_i maintains α charge and at least c spikes, so that all c spikes are removed from neuron σ_i and the β charge is sent to neuron σ_j by synapse (i, j).

At some point, if a rule from $R_{(i,j)}$ can be used by a synapse (i, j), the rule must be used. If several rules from $R_{(i,j)}$ can be used by a synapse, these rules are chosen nondeterministically. For example, there are two firing rules, $\alpha_1/a^c \longrightarrow a/\lambda;\beta_1$ and $\alpha_1/a^c \longrightarrow a/\lambda;\beta_2$, in a synapse, with $\alpha_1, \beta_1, \beta_2 \in \{-, 0, +\}$, $\beta_1 \neq \beta_2$. The synapse nondeterministically applies one of the rules. Meanwhile, when some rules associated with several synapses originating from the same neuron can be applied, all rules that are applied consume the same number of spikes from the same neuron. In each unit of time, only one rule on each synapse is used. At the system level, rules are used in parallel at different synapses.

In PSNRS P systems, the use of rules is determined by the polarity of neurons and the number of pulses located in neurons and no longer matches the regular expression by checking the number of pulses. More broadly, to use a rule, the total number of spikes inside the neuron should not be less than the number of spikes consumed by the rule. Moreover, the neurons send out not only spikes but also charges, even when using forgetting rules.

Specifically, when a neuron receives a charge from neighboring neurons, the changes in the electrical charges connected to the neuron proceed as follows:

(i) Two or more positive charges (+) and two or more negative charges (−) lead to one positive charge (+) and one negative charge(−), respectively; several neutral charges (0) lead to one neutral charge (0)

(ii) One positive (+) and one negative charge (−) cancel each other and give the neutral charge (0)

(iii) One positive (+) or one negative charge (−) cannot be changed by a neutral charge (0)

In this work, system Π is considered as a number generator. Usually, the time interval between the first two spikes output by the output neuron is used as the result of a computation. The number $t_2 - t_1$ is said to be computed by systems Π, denoted by $N_2(\Pi)$.

In addition, system Π can work in an accepting mode, where the output neuron is removed. A number n is introduced in the system, by introducing a sequence $10^{n-1}1$ in neuron σ_{in}. This number n is said to be accepted by system Π if the computation eventually halts. The set of numbers accepted by Π is denoted by $N_{\text{acc}}(\Pi)$.

We denote the family of all sets of numbers generated or accepted by PSNRS P systems by $N_\alpha^{\text{syn}}\text{PSNP}_m^n$, where the symbol $\alpha \in \{2, \text{acc}\}$ indicates the generating or accepting mode, syn represents rules on synapses, there are up to m neurons, and each neuron has up to n rules on its synapses. As usual, the indices m and n are replaced with $*$ when no bound is imposed on the corresponding parameter.

3. Computation Power of PSNRS P Systems

In this section, we mainly investigate the computation power of PSNRS P systems and prove that PSNRS P systems can generate all recursively enumerable sets of numbers.

We first briefly review the definition of register machines. The construction of a register machine is $M = (m, H, l_0, l_h, I)$, where m indicates the number of registers, H indicates the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and I indicates the set of instructions (each of which is precisely labeled by an individual label from H). Each instruction is one of the following forms: ADD instruction l_i: $(\text{ADD}(r), l_j, l_k)$ indicates that the register r is added 1, and then the present instruction labeled with l_i passes to the next instruction l_j or l_k (being chosen nondeterministically); SUB instruction l_i: $(\text{SUB}(r), l_j, l_k)$ shows that the register r is subtracted 1 (if the register r is nonzero), and the present instruction labeled with l_i passes to the instruction l_j, or else (if the register r is zero) the present instruction labeled with l_i passes to the instruction l_k.

It is known that register machines with three registers can precisely generate a family of sets of recursively enumerable natural numbers; in other words, it can characterize NRE [34].

Theorem 1. *$N_2^{syn}PSNP_*^2 = NRE$.*

Proof. We prove only the inclusion $\text{NRE} \subseteq N_2^{\text{syn}}\text{PSNP}_*^2$; the reverse inclusion can be invoked from the Church–Turing thesis. For this, a specific PSNRS P system Π is constructed to simulate a register machine M, and the register machine $M = \{m, H, l_0, l_h, I\}$ is considered. Each register of M is associated with a neuron of Π; we represent the number n contained in register r by $2n$ spikes located in σ_r, and the instruction l of H corresponds to a neuron σ_l. Initially, all neurons are empty; when neuron σ_l is active, the instruction labeled with l starts. When two spikes are received by neuron σ_{l_i}, system Π starts to simulate instruction l_i: $(\text{OP}(r), l_j, l_k)$ (OP denotes one of the operations ADD and SUB), and either neuron σ_{l_j} or σ_{l_k} receives two spikes and a neutral charge, one of which is activated. Then, the system starts to simulate the corresponding instruction. During the simulation of M, when neuron σ_{l_h}, which is associated with the halting label l_h of M, is activated, the computation halts. Without any loss of generality, all registers other than register 1 are empty in the halting configuration, and register 1 is never decremented during the computation. The time interval between the first two spikes output by the output neuron σ_{out} is used as the computation result.

The PSNRS P system Π consists of the following three components: ADD, SUB, and FIN modules, shown in Figures 1–3, respectively. □

In the following sections, we explain how these modules work.

ADD module: simulating an ADD instruction l_i: $(\text{ADD}(r), l_j, l_k)$.

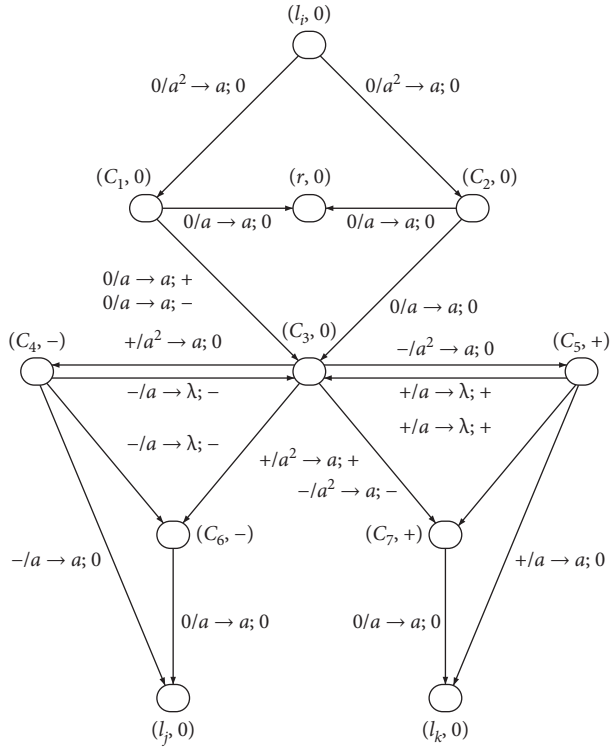

Figure 1: ADD module (simulating l_i: $(\mathrm{ADD}(r), l_j, l_k)$).

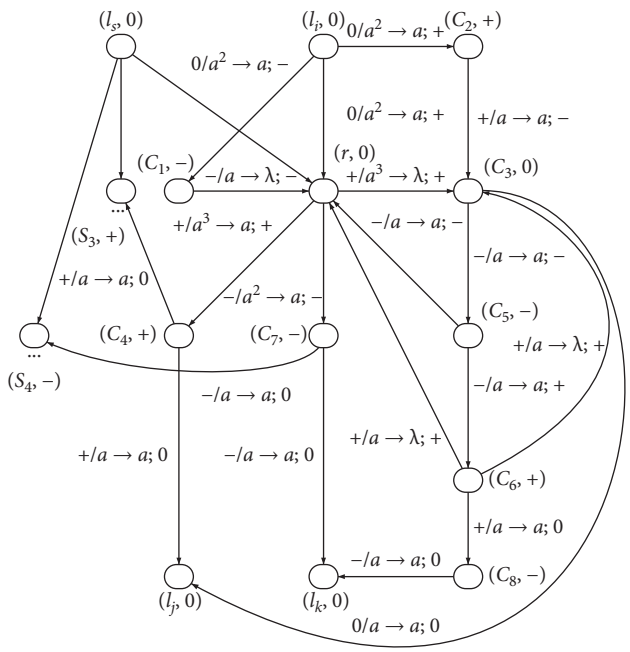

Figure 2: SUB module (simulating l_i: $(\mathrm{SUB}(r), l_j, l_k)$).

As shown in Figure 1, suppose that an ADD instruction l_i: $(\mathtt{ADD}(r), l_j, l_k)$ is simulated in step t, and neuron σ_{l_i} has a neutral charge and receives two spikes.

At step t, neuron σ_{l_i} receives two spikes, rule $0/a^2 \longrightarrow a; 0$ on synapses (l_i, C_1) and (l_i, C_2) is applied, and then neuron σ_{l_i} sends one spike and a neutral charge to auxiliary neurons σ_{c_1} and σ_{c_2}, respectively. At step $t+1$,

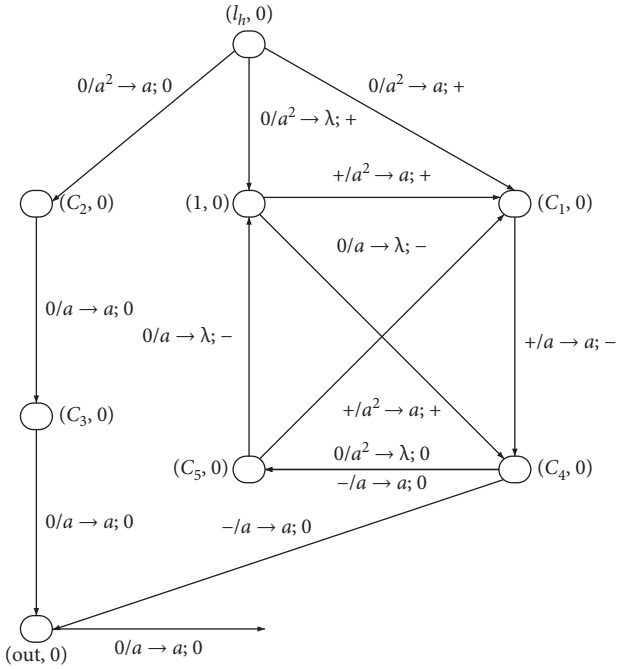

Figure 3: FIN module (ending the computation).

auxiliary neurons σ_{c_1} and σ_{c_2} both receive one spike and a neutral charge, the polarity of which remain the same. Rule $0/a \longrightarrow a; 0$ is used on synapses (C_1, r) and (C_2, r), and each of the neurons sends a spike to neuron σ_r. When the two spikes are received, the number of spikes in neuron σ_r is increased by two, simulating an increase in the number stored in register r by one. At the same time, neuron σ_{c_3} receives a spike from each of neurons σ_{c_1} and σ_{c_2}. Rule $0/a \longrightarrow a; 0$ is applied on synapse (C_2, C_3), and neuron σ_{c_2} sends a spike and a neutral charge to neuron σ_{c_3}. Then, one of the rules $0/a \longrightarrow a; +$ and $0/a \longrightarrow a; -$ on synapse (C_1, C_3) is nondeterministically chosen and applied, and neuron σ_{c_1} sends a spike to neuron σ_{c_3}. There are two possible cases depending on the polarity received by neuron σ_{c_3}.

Case I. At step $t+1$, if rule $0/a \longrightarrow a; +$ is used on synapse (C_1, C_3), then neuron σ_{c_3} receives a spike and positive charge from neuron σ_{c_1}, and the polarization of the neuron is changed from a neutral charge to a positive charge. At step $t+2$, neuron σ_{c_3} sends a spike to neuron σ_{c_4} via rule $+/a^2 \longrightarrow a; 0$ on synapse (C_3, C_4), and a spike is maintained in neuron σ_{c_4}. At the same step, rule $+/a^2 \longrightarrow a; +$ is applied on synapse (C_3, C_6), and neuron σ_{c_6} receives a spike and a positive charge. In this way, neuron σ_{c_6} accumulates a spike and changes to a neutral charge. At step $t+3$, neuron σ_{l_j} receives two spikes from neurons σ_{c_4} and σ_{c_6}, which both have a neutral charge. Meanwhile, by means of rule $-/a \longrightarrow \lambda; -$, neuron σ_{c_4} sends a negative charge to neurons σ_{c_3} and σ_{c_6}, so the polarizations of neurons σ_{c_3} and σ_{c_6} are changed back to their original state. At the same step, rule is used via synapse (C_4, l_j), and neuron σ_{c_4} sends a spike to neuron σ_{l_j}. In this way, two spikes are present in neuron σ_{l_j}, and system Π starts to simulate instruction l_j of M.

Case II. At step $t+1$, if rule $0/a \longrightarrow a;-$ can be enabled on synapse (C_1, C_3), a spike and a negative charge are sent to neuron σ_{c_3}. At step $t+2$, neuron σ_{c_5} receives a spike via synapse (C_3, C_5) via rule $-/a^2 \longrightarrow a;0$. At the same step, neuron σ_{c_3} sends a spike and negative charge to neuron σ_{c_7} along synapse (C_3, C_7) by rule $-/a^2 \longrightarrow a;-;$ afterwards, neuron σ_{c_7} has a neutral charge. At step $t+3$, rule $+/a \longrightarrow \lambda;+$ is applied on synapse (C_5, C_3), and neuron σ_{c_3} receives a positive charge and returns the original state. Similarly, neuron σ_{c_5} executes the same rule for neuron σ_{c_7} via synapse (C_5, C_7). Otherwise, neuron σ_{l_k} receives two spikes from neurons σ_{c_5} and σ_{c_7} by rules $0/a \longrightarrow a;0$ and $+/a \longrightarrow a;0$ via synapses (C_5, l_k) and (C_7, l_k). In this way, the system Π starts to simulate instruction l_k of M.

Therefore, the ADD module can correctly simulate ADD instructions: in neuron σ_r, the number of spikes is increased by two; meanwhile, one of two neurons σ_{l_j} and σ_{l_k} nondeterministically receives two spikes.

SUB module: simulating a SUB instruction l_i: $(\mathtt{SUB}(r), l_j, l_k)$.

In this section, we will describe the SUB module, as shown in Figure 2. Suppose that a SUB instruction l_i: $(\mathtt{SUB}(r), l_j, l_k)$ is simulated and neuron σ_{l_i} receives two spikes at step t. Initially, neuron σ_{l_i} fires, sending spikes to neuron σ_{c_2} and neuron σ_r using rule $0/a^2 \longrightarrow a;+$ on synapses (l_i, C_2) and (l_i, r). However, neuron σ_{c_2} maintains a positive charge, and neuron σ_r is changed to a positive charge. At the same step, rule $0/a^2 \longrightarrow a;-$ is applied on synapse (l_i, C_1), and neuron σ_{c_1} receives a spike from neuron σ_{l_i}. According to the number of spikes in neuron σ_r, the following two cases are considered.

Case I. At step $t+1$, neuron σ_r has $2n+1 (n \geq 0)$ spikes (corresponding to the number stored in register r, being n). In this way, neuron σ_r has a positive charge, rule $+/a^3 \longrightarrow \lambda;+$ is applied on synapse (r, C_3), and neuron σ_{c_2}, which has a positive charge, sends a spike and a negative charge to neuron σ_{c_3}. Note that neuron σ_{c_3} maintains a neutral charge when a positive charge meets a negative charge, resulting in a spike in neuron σ_{c_3}. Simultaneously, neuron σ_r fires, sending a spike to neuron σ_{c_4} via rule $+/a^3 \longrightarrow a;+$ on synapse (r, C_4). At the same step, neuron σ_{c_1} sends a negative charge to neuron σ_r, and the initial state of neuron σ_r is recovered to avoid inaccurate operation. At step $t+2$, neuron σ_{c_4} is activated by rule $+/a \longrightarrow a;0$ on synapse (C_4, l_j), sending a spike to neuron σ_{l_j}; meanwhile, neuron σ_{c_3} sends a spike to neuron σ_{l_j} via rule $0/a \longrightarrow a;0$ on synapse (C_3, l_j). Neuron σ_{l_j} receives two spikes, which indicates that system Π simulates the instruction l_j of M.

Case II. At step $t+1$, neuron σ_r has a spike (corresponding to the number stored in register r being 0). Then, a negative charge is sent to neuron σ_r via rule $-/a \longrightarrow \lambda;-$ on synapse (C_1, r), and neuron σ_r returns to the initial state when a positive charge meets a negative charge. At the same step, neuron σ_{c_2} is activated by rule $+/a \longrightarrow a,-$ on synapse (C_2, C_3), sending a spike and a negative charge to neuron σ_{c_3}. At step $t+2$, rule $-/a \longrightarrow a;-$ is enabled on synapse (C_3, C_5), and neuron σ_{c_3} sends a spike and a negative charge to neuron σ_{c_5}. At step $t+3$, neuron σ_{c_5} fires, sending a spike to neurons σ_r and σ_{c_6} by rules $-/a \longrightarrow a;-$ and $-/a \longrightarrow a;+$ on synapses (C_5, r) and (C_5, C_6). Then, there are two spikes in neuron σ_r with a neutral charge and a spike in neuron σ_{c_6}. At step $t+4$, rule $-/a^2 \longrightarrow a;-$ is applied via synapse (r, C_7), and a spike is sent to neuron σ_{c_7} from neuron σ_r. Simultaneously, neuron σ_{c_8} receives a spike from neuron σ_{c_6} by rule $+/a \longrightarrow a;0$ on synapse (C_6, C_8). Meanwhile, neuron σ_{c_6} sends a positive charge to neurons σ_r and σ_{c_3}, restoring their initial states. At step $t+5$, neuron σ_{l_k} receives two spikes from neurons σ_{c_7} and σ_{c_8} via rule $-/a \longrightarrow a;0$ on synapses (C_7, l_k) and (C_8, l_k). In this way, system Π simulates the instruction l_k.

The simulation of the SUB instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$ is correct: system Π starts with neuron σ_{l_i} having two spikes inside and ends with sending two spikes and a neutral charge to neuron σ_{l_j} (if the number located in register r is greater than 0, the register r is decreased by one) or sending two spikes and a neutral charge to neuron σ_{l_k} (if the number located in register r is 0).

Obviously, there is no interference between the ADD and SUB modules, but interferences exist between two SUB modules. If there are two SUB instructions l_s acting on register r, neurons σ_{c_4} and σ_{c_7} associated with register r send synapses to neurons σ_{s_3} and σ_{s_4}, respectively. In the SUB module associated with l_s $(l_s \neq l_i)$, when we simulate a SUB instruction l_i: $(\mathtt{SUB}(r), l_j, l_k)$, all neurons except neurons σ_{s_3} and σ_{s_4} receive no spikes and charges.

It is important to note that the synapses and their rules associated with neurons σ_{s_3} and σ_{s_4} are not shown in Figure 2; for example, neurons σ_{s_3} and σ_{s_4} exist between any two subtraction modules, and the rule $+/a \longrightarrow \lambda;0$ is present on the synapse of such two neurons, respectively. Neurons σ_{s_3} and σ_{s_4} only consume spikes and restore neutral charge, not affecting other neurons' states. Moreover, the synapses of neurons σ_{s_3} and σ_{s_4} may be connected to any neuron in the environment or neurons related to SUB instruction l_s; that is to say, the synaptic connections between neuron σ_{s_3} (or σ_{s_4}) and other neurons cannot be described in Figure 2.

When $n>0$ spikes are present in neuron r, neuron σ_{c_4} sends a spike and a neutral charge to neuron σ_{s_3}; therefore, neuron σ_{s_3} fires, rule $+/a \longrightarrow \lambda;0$ (not shown in Figure 2) is enabled, then one spike a is forgotten, and neuron σ_{s_3} returns to the initial state. If no spike exists in neuron r, neuron σ_{s_4} receives a spike and a neutral charge from neuron σ_{c_7}, rule $+/a \longrightarrow \lambda;0$ is enabled, and one spike a is forgotten. Next, neurons σ_{s_3} and σ_{s_4} return to their initial state with respect to the initial number of spikes. Consequently, there is no interference between the SUB modules.

FIN module: outputting the result of the computation.

As shown in Figure 3, the FIN module is constructed. Assume that the computation halts; in other words, the halt instruction l_h is reached and the result of the computation is stored in register 1 (register 1 contains n spikes). Neuron σ_{l_h} receives two spikes and a neutral charge at step t, rule $0/a^2 \longrightarrow a;0$ on synapse (l_h, C_2) is activated, and neuron σ_{l_h} sends a spike and a neutral charge to neuron σ_{c_2}. At that

moment, neuron σ_{l_h} fires, sending a positive charge to neuron σ_1 via rule $0/a^2 \longrightarrow \lambda; +$ on synapse $(l_h, 1)$. Meanwhile, neuron σ_{c_1} receives a spike and a positive charge from neuron σ_{l_h} by rule $0/a^2 \longrightarrow a; +$ via synapse (l_h, C_1). At step $t+1$, neuron σ_{c_2} sends a spike and a neutral charge to neuron σ_{c_3} via synapse (C_2, C_3) by rule $0/a \longrightarrow a; 0$. Simultaneously, neuron σ_1 sends a spike and a positive charge to neurons σ_{c_1} and σ_{c_4} via rule $+/a^2 \longrightarrow a; +$. Meanwhile, rule $+/a \longrightarrow a; -$ on synapse (C_1, C_4) is enabled, and neuron σ_{c_4} receives a spike and a negative charge from neuron σ_{c_1}.

At that moment, a positive charge and a negative charge meet in neuron σ_{c_4}, which therefore has a neutral charge and two spikes. At step $t+2$, rule $0/a \longrightarrow a; 0$ on synapse (C_3, out) is applied, and a spike is sent from neuron σ_{out}. Meanwhile, neuron σ_{c_4} fires, sending no spike and a neutral charge to neuron $.\sigma_{c_5}$ via rule $0/a^2 \longrightarrow \lambda; 0$ on synapse (C_4, C_5). At step $t+3$, rule $0/a \longrightarrow a; 0$ is applied, and neuron σ_{out} sends the first spike to the environment. At the same time, neuron σ_{c_4} receives two spikes from neurons σ_1 and σ_{c_1}.

From step $t+3$ to step $t+n+1$, the spike of neuron σ_1 is exhausted by rule $0/a^2 \longrightarrow \lambda; 0$ on synapse (C_4, C_5). At step $t+n+1$, neuron σ_{c_4} receives a spike and negative charge from neuron σ_{c_1}. Then, neuron σ_{c_4}, which now has a negative charge, sends a spike and a neuron charge to neurons σ_{out} and σ_{c_5} by rule $-/a \longrightarrow a; 0$ on synapses (C_4, out) and (C_4, C_5) at step $t+n+2$. At step $t+n+3$, neuron σ_{c_5} fires, sending a negative charge via synapses $(C_5, 1)$ and (C_5, C_1) and restoring the initial state of charge. Meanwhile, neuron σ_{out} sends the second spike to the environment by rule $0/a \longrightarrow a; 0$. Hence, the interval at which two spikes are sent to the environment by the system Π is $(t+n+3)-(t+3) = n$, where n is exactly the number stored in register 1, i.e., corresponding to the result computed by system. Therefore, the register machine M is correctly simulated by the system Π, $N(M) = N_2(\Pi)$.

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

Proof. A PSNRS P system Π', working in the accepting mode, is constructed to simulate a deterministic register machine $M = (m, H, l_0, l_h, I)$. The system Π' contains an INPUT module, a deterministic ADD module, and a SUB module. The INPUT module is shown in Figure 4. Spike train $10^{n-1}1$ is introduced into Π' by means of neuron σ_{in}. In this system, the time interval of the first two spikes introduced by the INPUT module is used as the computing result.

At step t, we suppose that neuron σ_{in} receives the first spike from the environment. Neuron σ_{in} fires, sending a spike and neutral charge to neuron σ_{in_1} via rule $0/a \longrightarrow a; 0$ on synapse (in, in_1). At the same step, rule $0/a \longrightarrow a; -$ is applied on synapses (in, in_5) and (in, in_6). A spike is sent to neurons σ_{in_5} and σ_{in_6}, and their polarity is changed to a negative charge. Similarly, neurons σ_{in_4} and σ_{in_7} receive one spike and a neutral charge from neuron σ_{in} via rules $0/a \longrightarrow a; 0$ (corresponding to synapses (in, in_4) and (in, in_7)). At step $t+1$, neurons σ_{in_2} and σ_{in_3} receive a spike and neutral charge from neuron σ_{in_1}, respectively. Meanwhile, neuron σ_{in_5} fires, rule $-/a \longrightarrow a; 0$ is applied on synapses $(\text{in}_5, \text{in}_6)$ and $(\text{in}_5, 1)$, and neurons σ_{in_6} and σ_1 receive a spike and a neutral charge from neuron σ_{in_5}, respectively. Similarly, neurons σ_{in_5} and σ_1 receive a spike and a neutral charge by rule $-/a \longrightarrow a; 0$ on synapses $(\text{in}_6, \text{in}_5)$ and $(\text{in}_6, 1)$. From step $t+1$ on, neurons σ_{in_5} and σ_{in_6} exchange one spike with each other, and neuron σ_1 receives two spikes in each step. In this case, at step $t+n-1$, neuron σ_{in} receives the second spike from the environment. Then, neurons σ_{in_4}, σ_{in_5}, σ_{in_6}, and σ_{in_7} receive the second spike and accumulate two spikes. Note that neuron σ_1 still receives spikes at step $t+n$, and there are $2n$ spikes in neuron σ_1, which represents the number stored in register 1 of M.

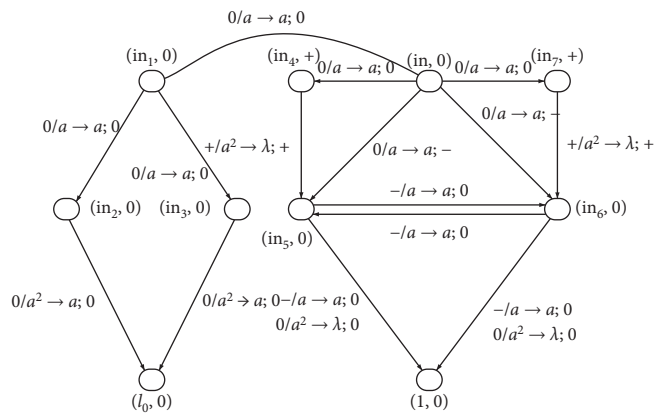

Figure 4: INPUT module of Π'.

At the same step, neurons σ_{in_4} and σ_{in_7} send a positive charge to neurons σ_{in_5} and σ_{in_6}, respectively, restoring their original states of charge. Meanwhile, neurons σ_{in_2} and σ_{in_3} have spikes according to rule $0/a \longrightarrow a; 0$ on synapses $(\text{in}_1, \text{in}_2)$ and $(\text{in}_1, \text{in}_3)$. At step $t+n+1$, the spikes in neurons σ_{in_5} and σ_{in_6} are depleted by rule $0/a^2 \longrightarrow \lambda; 0$ on synapses $(\text{in}_5, 1)$ and $(\text{in}_6, 1)$.

At the same step, neuron σ_{l_0} receives two spikes and a neutral charge from neurons σ_{in_2} and σ_{in_3} by rule $0/a^2 \longrightarrow a; 0$. In this way, neuron σ_{l_0} receives two spikes and the system simulates the initial instruction l_0 of M.

For a deterministic ADD instruction of the form l_i: $(\text{ADD}(r), l_j)$, the corresponding ADD module shown in Figure 5 is simpler than that shown in Figure 1. Hence, we do not consider the details here.

In system Π', the SUB module remains unchanged as shown in Figure 2, and neuron σ_{l_h} remains in the system; however, the FIN module is removed. There is no rule in neuron σ_{l_h}. When neuron σ_{l_h} receives two neurons and no rules are available, the computation stops. The computing result introduced is $(t+n+1)-(t+1) = n$, which means that the number accepted by system Π' is n. The operation process of INPUT module is shown in Table 1.

According to the above description, the register machine M working in accepting mode can be correctly simulated by PSNRS P systems. Hence, the Theorem holds. □

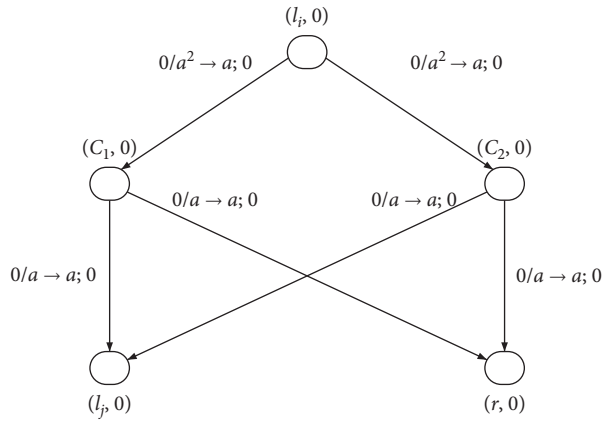

Figure 5: ADD module of Π'.

4. A Small Universal PSNRS P System

In this section, a small universal PSNRS P system of computing device is constructed. Generally, a register machine M is used to compute a Turing computable function f: $\mathbf{N}^k \longrightarrow \mathbf{N}$ in the following way: arguments $n_1, n_2, \ldots, n_k$ are introduced by the specified registers $r_1, r_2, \ldots, r_k$ (representing the first k registers). We begin with the first instruction with label l_0 and stop with the halt instruction with label l_h (if we stop), and then the value of the function is stored in a specific register r_t; beyond that, all other registers are empty. For further information about universal register machines for computable functions, readers can refer to [34].

In the following proof of the universal result, we use a specific universal register machine mentioned in [34], as shown in Figure 6. The universal register machine is of the construct $M_u = (8, H, l_0, l_h, I)$, where there are 8 registers, numbered from 0 to 7, and 23 instructions; the last instruction is the halting one. As defined in [34], the input numbers are introduced in registers 1 and 2, and the result is stored in register 0 when the machine M_u halts.

According to the previous description, subtraction instructions are not allowed on the registers where the results are stored, but register 0 of M_u is subject to SUB instructions. Therefore, register machine M_u is modified into M_u': a register with label 8 is added, and the halting instruction l_h of M_u is replaced by the following instructions:

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

In this way, the universal register machine M_u' has 9 registers, 24 ADD and SUB instructions, and 25 labels. The result of the computation is stored in register 8, which is never decremented during the computation. Furthermore, the register machine M_u' can be considered deterministic, and, without losing Turing completeness, the ADD instructions l_i: $(\mathrm{ADD}(r), l_j, l_k)$ having $l_j = l_k$ is denoted by the form l_i: $(\mathrm{ADD}(r); l_j)$.

Table 1: Simulation for the INPUT module.

Step	Neurons active	Rules executed	Synapses
t	$\mathrm{in}, 0, a$	$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_1)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_4)$
		$0/a \longrightarrow a; -$	$(\mathrm{in}, \mathrm{in}_5)$
		$0/a \longrightarrow a; -$	$(\mathrm{in}, \mathrm{in}_6)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_7)$
$t+1$	$\mathrm{in}_1, 0, a$	$0/a \longrightarrow a; 0$	$(\mathrm{in}_1, \mathrm{in}_2)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}_1, \mathrm{in}_3)$
	$\mathrm{in}_5, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, \mathrm{in}_6)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, \mathrm{in}_5)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, 1)$
$t+2$	$\mathrm{in}_5, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, \mathrm{in}_6)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, \mathrm{in}_5)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, 1)$
$t+3$	$\cdots$	$\cdots$	$\cdots$
$t+4$	$\mathrm{in}_5, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, \mathrm{in}_6)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, \mathrm{in}_5)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, 1)$
$t+n-1$	$\mathrm{in}, 0, a$	$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_1)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_4)$
		$0/a \longrightarrow a; -$	$(\mathrm{in}, \mathrm{in}_5)$
		$0/a \longrightarrow a; -$	$(\mathrm{in}, \mathrm{in}_6)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}, \mathrm{in}_7)$
	$\mathrm{in}_5, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, \mathrm{in}_6)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, -, a$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, \mathrm{in}_5)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, 1)$
$t+n$	$\mathrm{in}_1, 0, a$	$0/a \longrightarrow a; 0$	$(\mathrm{in}_1, \mathrm{in}_2)$
		$0/a \longrightarrow a; 0$	$(\mathrm{in}_1, \mathrm{in}_3)$
	$\mathrm{in}_5, -, a^2$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, \mathrm{in}_6)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, -, a^2$	$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, \mathrm{in}_5)$
		$-/a \longrightarrow a; 0$	$(\mathrm{in}_6, 1)$
	$\mathrm{in}_4, +, a$	$+/a \longrightarrow \lambda; +$	$(\mathrm{in}_4, \mathrm{in}_5)$
	$\mathrm{in}_7, +, a$	$+/a \longrightarrow \lambda; +$	$(\mathrm{in}_7, \mathrm{in}_6)$
$t+n+1$	$\mathrm{in}_2, 0, a^2$	$0/a^2 \longrightarrow a; 0$	(in_1, l_0)
	$\mathrm{in}_3, 0, a^2$	$0/a^2 \longrightarrow a; 0$	(in_1, l_0)
	$\mathrm{in}_5, 0, a^2$	$0/a^2 \longrightarrow \lambda; 0$	$(\mathrm{in}_5, 1)$
	$\mathrm{in}_6, 0, a^2$	$0/a^2 \longrightarrow \lambda; 0$	$(\mathrm{in}_6, 1)$

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

Figure 6: The universal register machine M_u.

Theorem 3. *There is a universal PSNRS P system with 151 neurons for computable functions.*

Proof. A PSNRS P system Π' is constructed to simulate the universal register machine M_u', as shown in Figure 7. The system Π' consists of seven modules: ADD, ADD-ADD, SUB, ADD-SUB, SUB-ADD, INPUT, and OUTPUT. The ADD module (resp., SUB module) is applied to simulate the ADD instruction (resp., SUB instruction) of M_u'; in a similar way, each module is applied to simulate its corresponding instruction. However, the INPUT module is used to introduce a spike train from the environment, and the OUTPUT module is used to output the computation result.

In system Π', each register r corresponds to a neuron σ_r, and the number saved in register r is encoded by the number of spikes stored in neuron σ_r. Specifically, register r having the number n ($n \geq 0$) is equivalent to neuron σ_r containing $2n$ spikes. Moreover, a neuron σ_{l_i} in system Π' is associated with each instruction l_i in M_u'. When neuron σ_{l_i} holds two spikes, the rules on synapses are enabled, which means system Π' simulates the instruction l_i. Until two spikes and a neutral charge are received by neuron $\sigma_{l_h'}$, the computation in M_u' is simulated by system Π'. The number of spikes emitted from system Π' into the environment is exactly the result computed by M_u'.

All modules are presented in graphical form, indicating the neurons and synapses with the associated sets of rules. In the initial configuration, there is no spike in each neuron.

The INPUT module, shown in Figure 8, introduces $2g(x)$ and $2y$ spikes into neurons σ_1 and σ_2 by reading the spike train $10^{g(x)-3}10^{y-2}1$, respectively. The module works as follows.

Initially, neuron σ_{in} receives one spike from the environment. Neuron σ_{in} fires, rule $0/a \longrightarrow a; 0$ is applied, and a spike and a neutral charge are sent to neurons $\sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}, \sigma_{c_7}, \sigma_{c_8}, \sigma_{c_9}$. Afterwards, neuron σ_{c_3} fires, sending a spike and a negative charge to neurons σ_{c_5} and σ_{c_6} by using rules $0/a \longrightarrow a; -$ via synapses (C_3, C_5) and (C_3, C_6). The neuron σ_{c_4} holds a spike, and no rule is used. Then, neurons σ_{c_5} and σ_{c_6} have neutral charge, and the rules $0/a \longrightarrow a; 0$ on synapses (C_5, C_6) and (C_6, C_5) can be enabled. From that moment on, neurons σ_{c_5} and σ_{c_6} exchange a spike and a neutral charge with each other until the neuron σ_{c_4} receives the second spikes. After $g(x) - 2$ step, neuron σ_{c_4} receives second spike. One step later, the neuron σ_{c_4} receives a spike and neutral charge. Subsequently, rules $+/a^2 \longrightarrow \lambda; +$ on synapses (C_4, C_5) and (C_4, C_6) are applied, and neurons σ_{c_5} and σ_{c_6} are changed to positive charge. Rules $+/a^2 \longrightarrow \lambda; 0$ can be enabled, removing two spikes in each of the neurons σ_{c_5} and σ_{c_6}. At that moment, no rules associated with neuron σ_1 can be used. In this way, $2g(x)$ spikes are introduced in neuron σ_1.

When the second spike arrives in neurons σ_{c_8} and σ_{c_9}, rules $+/a^2 \longrightarrow a; -$ on synapses (C_8, C_9) and (C_9, C_8) are applied, and then each one receives one spike and one negative charge; hence, they become neutral and contain one spike. From the next moment on, rules $0/a \longrightarrow a; 0$ are used, and neurons σ_{c_8} and σ_{c_9} emit a spike and a neutral charge to each other, in each step, until the third spike arrives in neurons σ_{c_7}. After $y - 1$ step, neuron σ_{c_7} gets the third spike, and rules $+/a^3 \longrightarrow \lambda; -$ on synapses (C_7, C_8) and (C_7, C_9) can be applied. The neurons σ_{c_8} and σ_{c_9} have negative charge, and no rules are applied. At the same time, rules $+/a^3 \longrightarrow \lambda; +$ via synapses (C_7, C_4), (C_7, C_5), and (C_7, C_6) are applied; neurons σ_{c_4}, σ_{c_5}, and σ_{c_6} still keep positive charge; and the extra spikes will be consumed by rules $+/a^2 \longrightarrow \lambda; +$ on synapse (C_4, C_5) and $+/a^2 \longrightarrow \lambda; 0$ via synapses (C_5, C_6), (C_6, C_5). In this way, $2y$ spikes have been loaded in neuron σ_2. With a third spike into neurons σ_{c_1} and σ_{c_2}, this two neurons fire, rules $0/a^3 \longrightarrow a; 0$ on synapses (C_1, l_0) and (C_2, l_0) can be enabled, the neuron l_0 receives two spikes and a neutral charge, and the system starts to simulate the initial instruction l_0 of M_u'.

For ADD and SUB modules, we will follow the ADD module in accepting mode as shown in Figure 5 and the SUB module in generating mode as shown in Figure 2.

When neuron $\sigma_{l_h'}$ receives two spikes, that is, the instruction l_h' is reached, the result of computation is store in register 8. The tasks of outputting the result is carried out by the OUTPUT module shown in Figure 9.

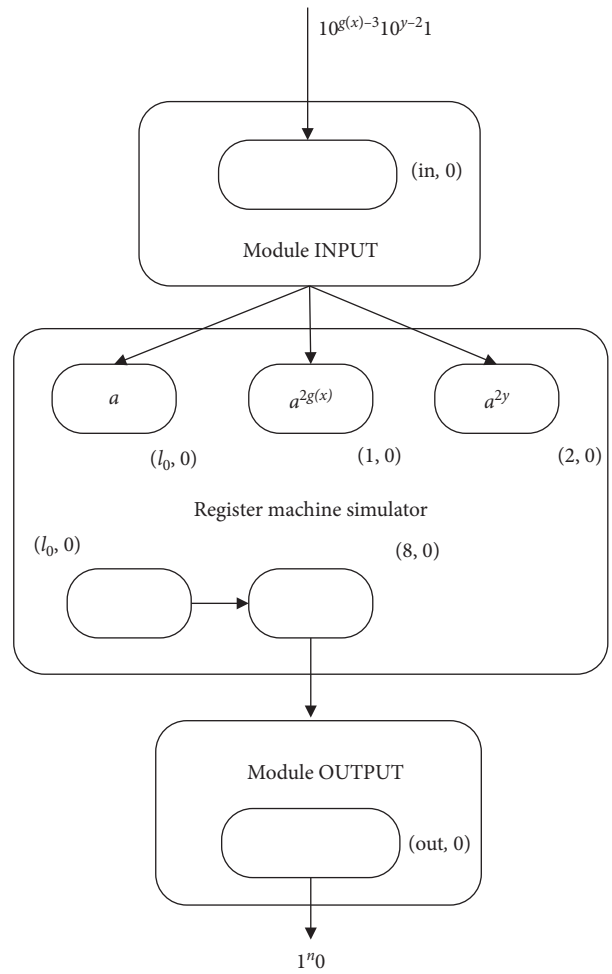

Figure 7: The general design of the universal PSNRS P systems.

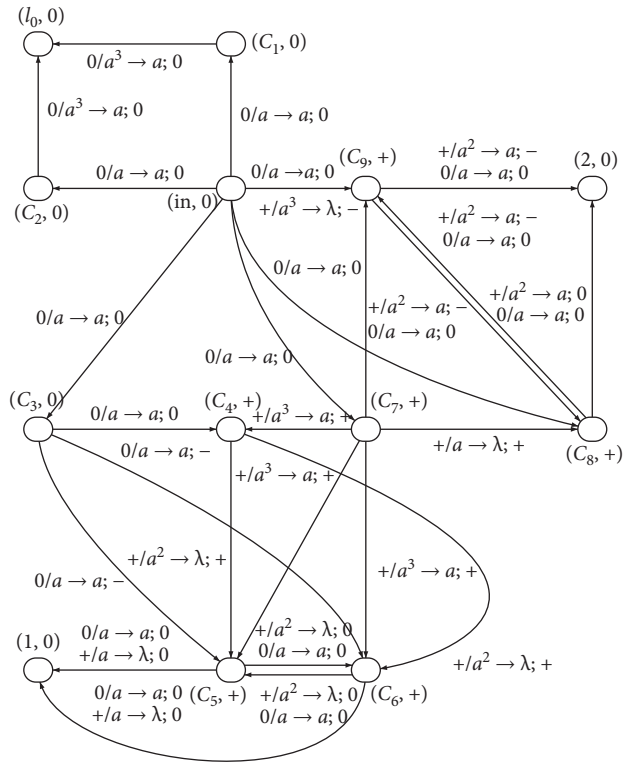

Figure 8: INPUT module of $\Pi\prime$.

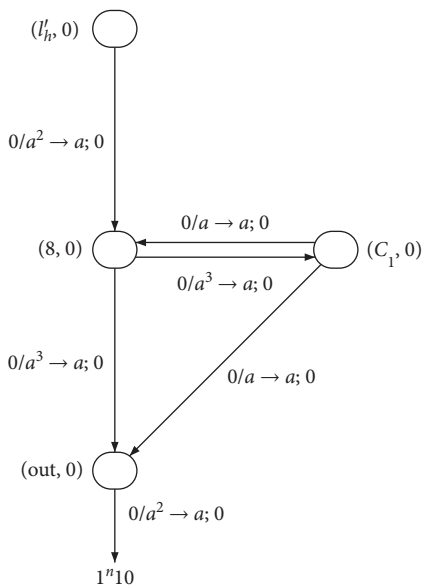

Figure 9: OUTPUT module of $\Pi\prime$.

Assume that, at step t, the neuron $\sigma_{l'_h}$ gets two spikes, rule $0/a^2 \longrightarrow a; 0$ on synapse $(l'_h, 8)$ is used, and neuron σ_8 receives a spike and a neutral charge. Then, there are $2n + 1$ spikes in neuron σ_8, it fires, using rules $0/a^3 \longrightarrow a; 0$ on synapses $(8, \text{out})$ and $(8, C_1)$, and a spike and one neutral charge are sent to neurons σ_{out} and σ_{c_1}. Then, neuron σ_{c_1} sends a spike and neutral charge to neurons σ_8 and σ_{out}, so neuron σ_{out} emits a spike and neutral charge out by rule $0/a^2 \longrightarrow a; 0$, until n spikes are sent out. Note that neuron σ_8 and neuron σ_{c_1} exchange a spike and neutral charge with each other, ensuring that neuron σ_{out} can emit all the results of computation for system M'_u.

Therefore, according to the above INPUT module, deterministic ADD module, SUB module, and OUTPUT module, we have used the following:

(i) 9 neurons for 9 registers

(ii) 25 neurons for 25 labels

(iii) 8×14 neurons for 14 SUB instructions

(iv) 2×10 neurons for 10 ADD instructions

(v) 10 neurons in the INPUT module

(vi) 2 neurons in the OUTPUT module

All these come to a total of 178 neurons.

This number can be slightly decreased by some code optimization, exploring some particularities of the register machine M'_u.

For instance, the sequence of ADD instructions

$$\begin{aligned} l_{17}&: (\text{ADD}(2), l_{21}), \\ l_{21}&: (\text{ADD}(3), l_{18}) \end{aligned} \tag{3}$$

without any other instruction addressing the label l_{21}, can be simulated by the module shown in Figure 10. Then, three neurons associated with label l_{21} can be saved.

There are also two pairs of ADD-SUB instructions:

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

Each sequence of ADD-SUB instructions,

$$\begin{aligned} l_i&: \left(\text{ADD}\left(r'\right), l_g\right), \\ l_g&: \left(\text{SUB}\left(r''\right), l_j, l_k\right), \end{aligned} \tag{5}$$

can be simulated by the ADD-SUB module shown in Figure 11.

In this way, we save the 6 neurons associated with labels l_6 and l_{10}.

A similar operation is possible for the following six sequences of SUB-ADD instructions:

$$\begin{aligned} l_0&: (\text{SUB}(1), l_1, l_2), \\ l_1&: (\text{ADD}(7), l_0), \\ l_4&: (\text{SUB}(6), l_5, l_3), \\ l_5&: (\text{ADD}(5), l_6), \\ l_6&: (\text{SUB}(7), l_7, l_8), \\ l_7&: (\text{ADD}(1), l_4), \\ l_8&: (\text{SUB}(6), l_9, l_0), \\ l_9&: (\text{ADD}(6), l_1 0), \\ l_{14}&: (\text{SUB}(5), l_{16}, l_{17}), \\ l_{16}&: (\text{ADD}(4), l_{11}), \\ l_h&: (\text{SUB}(1), l_{22}, l'_h), \\ l_{22}&: (\text{ADD}(8), l_h). \end{aligned} \tag{6}$$

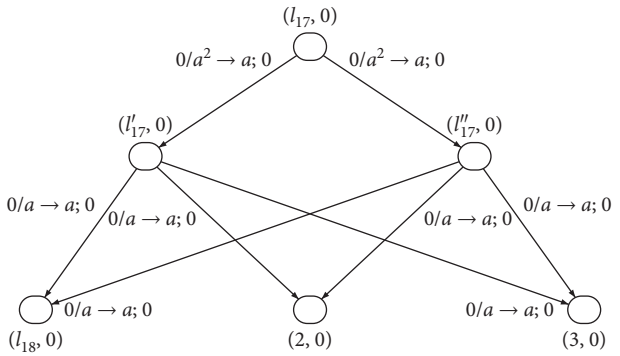

Figure 10: ADD-ADD module of $\Pi\prime$.

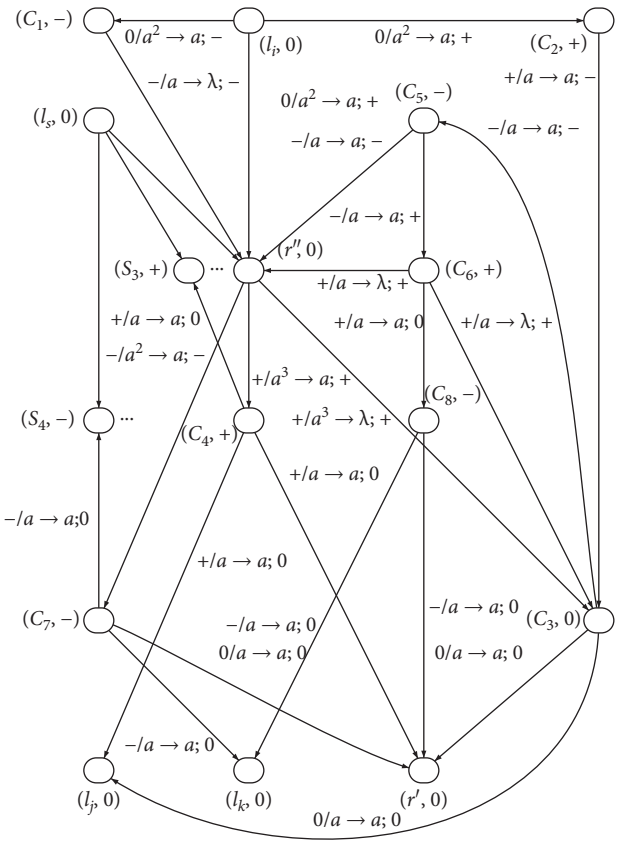

Figure 11: The module simulating consecutive ADD-SUB instructions.

Each sequence of ADD-SUB instructions,

$$\begin{aligned} l_g &: \left(\mathrm{SUB}\left(r''\right), l_j, l_k\right), \\ l_i &: \left(\mathrm{ADD}\left(r'\right), l_g\right), \end{aligned} \tag{7}$$

can be simulated by the SUB-ADD module given in Figure 12. In this way, we save 18 neurons associated with labels l_1, l_5, l_7, l_9, l_{16}, l_{22}.

Therefore, by using the ADD-ADD module, ADD-SUB module, and SUB-ADD module, we can totally save 27 neurons (3(ADD-ADD) + 6(ADD-SUB) + 18(SUB-ADD)). Then, the number of neurons can be decreased from 178 to 151. □

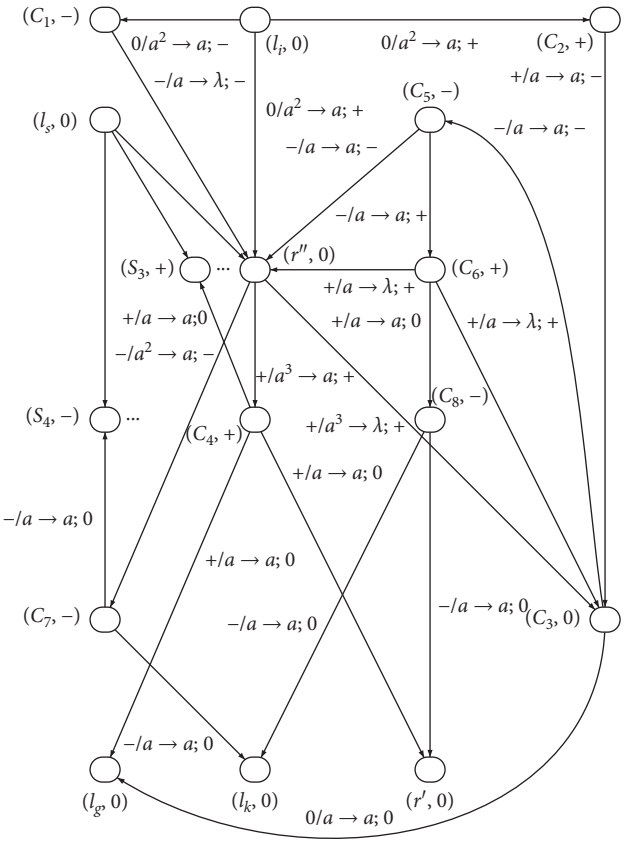

Figure 12: The module simulating consecutive SUB-ADD instructions.

5. Conclusions and Remarks

A variant of SN P systems, called SN P systems with polarizations and rules on synapses (PSNRS P systems), is proposed. We prove that as number generating devices and accepting devices, SN P systems with polarizations and rules on synapses are Turing universal. Moreover, a small universal system with 151 neurons as computing function device is given. Compared with the small universal SN P systems with polarizations proposed in [47], by moving rules to synapses, 13 neurons were reduced.

In the proof of Theorems 1 and 2, only standard spiking rules are used. It is interesting to use extended rules to construct universal systems with fewer neurons. In this work, PSNRS P systems are used as number generating or accepting devices. It is of interest to investigate the computation power of PSNRS P systems as language generators or to control language generators.

In general, some open problems about PSNRS P systems may be considered based on the results obtained in this work; for instance, it remains open whether the results about the universality of PSNRS P systems still hold with fewer neurons. The PSNRS P systems proved to be universal when using both spiking rules and forgetting rules; it is challenging to investigate whether PSNRS P systems are still universal when using only one type of rules. In this work, the systems work in the synchronous mode. It is of interest to study the

computation power of PSNRS P systems working in asynchronous mode [51] or local synchronous mode [52].

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work of S. Jiang was supported by the National Natural Science Foundation of China (61902360 and 61772214) and the Foundation of Young Key Teachers from University of Henan Province (2019GGJS131). The work of Y. Wang was supported by the National Key R&D Program of China for International S&T Cooperation Projects (2017YFE0103900), the Joint Funds of the National Natural Science Foundation of China (U1804262), and the State Key Program of National Natural Science Foundation of China (61632002). The work of F. Xu was supported by the National Natural Science Foundation of China (61502186), China Postdoctoral Science Foundation (2016M592335), and the Fundamental Research Funds for the Central Universities (HUST: 2019kfyXMBZ056).

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[2] A. Leporati, L. Manzoni, G. Mauri, A. E. Porreca, and C. Zandron, "Monodirectional P systems," *Natural Computing*, vol. 15, no. 4, pp. 551–564, 2016.

[3] G. Zhang, M. J. Pérez-Jiménez, and M. Gheorghe, *Real-Life Applications with Membrane Computing*, Springer-Verlag, Berlin, Germany, 2017.

[4] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón, "Tissue P systems," *Theoretical Computer Science*, vol. 296, no. 2, pp. 295–326, 2003.

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

[6] X. Zhang, L. Pan, and A. Paun, "On the Universality of Axon P systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2816–2829, 2015.

[7] S. Jiang, Y. Wang, J. Xu, and F. Xu, "The computational power of cell-like P systems with Symport/Antiport rules and promoters," *Fundamenta Informaticae*, vol. 164, no. 2-3, pp. 207–225, 2019.

[8] G. Ciobanu and A. Resios, "Computational complexity of simple P systems," *Fundamenta Informaticae*, vol. 87, no. 1, pp. 49–59, 2008.

[9] G. Păun and R. Păun, "Membrane computing and economics: numerical P systems," *Fundamenta Informaticae*, vol. 73, no. 1-2, pp. 213–227, 2006.

[10] A. Leporati, A. E. Porreca, C. Zandron, and G. Mauri, "Improved Universality results for parallel enzymatic numerical P systems," *International Journal of Unconventional Computing*, vol. 9, no. 5-6, pp. 385–404, 2013.

[11] G. Zhang, M. Gheorghe, and C. Wu, "A quantum-inspired evolutionary algorithm based on P systems for Knapsack problem," *Fundamenta Informaticae*, vol. 87, no. 1, pp. 93–116, 2008.

[12] Y. Rogozhin, A. Alhazov, L. Burtseva, S. Cojocaru, A. Colesnicov, and L. Malahov, "Solving problems in various domains by hybrid models of high performance computations," *Computer Science Journal of Moldova*, vol. 22, no. 1, pp. 3–20, 2014.

[13] A. Leporati, L. Manzoni, G. Mauri, A. E. Porreca, and C. Zandron, "Characterizing PSPACE with shallow non-confluent P systems," *Journal of Membrane Computing*, vol. 1, no. 2, pp. 75–84, 2019.

[14] R. A. B. Juayong and H. N. Adorna, "On simulating cooperative transition P systems in evolution-communication P systems with energy," *Natural Computing*, vol. 17, no. 2, pp. 333–343, 2018.

[15] K. Chen, J. Wang, Z. Sun, J. Luo, and T. Liu, "Programmable logic controller stage programming Using spiking neural P systems," *Journal of Computational and Theoretical Nanoscience*, vol. 12, no. 7, pp. 1292–1299, 2015.

[16] G. Păun, *The Oxford Handbook of Membrane Computing*, Oxford University Press, New York, NY, USA, 2010.

[17] M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1967.

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

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

[20] F. G. C. Cabarle, H. N. Adorna, and M. J. Pérez-Jiménez, "Sequential spiking neural P systems with structural plasticity based on max/min spike number," *Neural Computing and Applications*, vol. 27, no. 5, pp. 1337–1347, 2016.

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

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

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

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

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

[26] H. Peng, B. Li, J. Wang et al., "Spiking neural P systems with inhibitory rules," *Knowledge-Based Systems*, vol. 188, Article ID 105064, pp. 1–10, 2020.

[27] L. Pan, G. Păun, and G. Zhang, "Spiking neural P systems with communication on request," *International Journal of Neural Systems*, vol. 2, pp. 168–179, 2016.

[28] H. Chen, R. Freund, M. Ionescu, G. Păun, and M. J. Pérez-Jiménez, "On string languages generated by spiking neural P systems," *Fundamenta Informaticae*, vol. 75, no. 1–4, pp. 141–162, 2007.

[29] X. Zeng, L. Xu, X. Liu, and L. Pan, "On languages generated by spiking neural P systems with weights," *Information Sciences*, vol. 278, pp. 423–433, 2014.

[30] T. Wu, Z. Zhang, and L. Pan, "On languages generated by cell-like spiking neural P systems," *IEEE Transactions on NanoBioscience*, vol. 15, no. 5, pp. 455–467, 2016.

[31] R. T. A. de la Cruz, F. G. Cabarle, and H. N. Adorna, "Generating context-free languages using spiking neural P systems with structural plasticity," *Journal of Membrane Computing*, vol. 1, no. 3, pp. 161–177, 2019.

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

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

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

[35] Y. Rogozhin, "Small universal turing machines," *Theoretical Computer Science*, vol. 168, no. 2, pp. 215–240, 1996.

[36] T. Song, P. Zheng, M. L. Dennis Wong, and X. Wang, "Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control," *Information Sciences*, vol. 372, pp. 380–391, 2016.

[37] D. Díaz-Pernil, F. Peña-Cantillana, and M. A. Gutiérrez-Naranjo, "A parallel algorithm for skeletonizing images by using spiking neural P systems," *Neurocomputing*, vol. 115, pp. 81–91, 2013.

[38] H. Peng, J. Wang, M. J. Pérez-Jiménez, H. Wang, J. Shao, and T. Wang, "Fuzzy reasoning spiking neural P system for fault diagnosis," *Information Sciences*, vol. 235, pp. 106–116, 2013.

[39] T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, and M. J. Pérez-Jiménez, "Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems," *IEEE Transactions on Power Systems*, vol. 30, no. 3, pp. 1182–1194, 2015.

[40] H. Rong, K. Yi, G. Zhang, J. Dong, P. Paul, and Z. Huang, "Automatic implementation of fuzzy reasoning spiking neural P systems for diagnosing faults in complex power systems," *Complexity*, vol. 2019, Article ID 2635714, 16 pages, 2019.

[41] G. Zhang, H. Rong, F. Neri, and M. J. Pérez-Jiménez, "An optimization spiking neural P system for approximately solving combinatorial optimization problems," *International Journal of Neural Systems*, vol. 24, no. 5, pp. 1–16, 2014.

[42] G. Zhang, M. Gheorghe, L. Pan, and M. J. Pérez-Jiménez, "Evolutionary membrane computing: a comprehensive survey and new results," *Information Sciences*, vol. 279, pp. 528–551, 2014.

[43] A. B. Pavel and C. Buiu, "Using enzymatic numerical P systems for modeling mobile robot controllers," *Natural Computing*, vol. 11, no. 3, pp. 387–393, 2012.

[44] C. Buiu, C. Vasile, and O. Arsene, "Development of membrane controllers for mobile robots," *Information Sciences*, vol. 187, pp. 33–51, 2012.

[45] X. Wang, G. Zhang, F. Neri et al., "Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots," *Integrated Computer-Aided Engineering*, vol. 23, no. 1, pp. 15–30, 2016.

[46] C. Buiu and A. G. Florea, "Membrane computing models and robot controller design, current results and challenges," *Journal of Membrane Computing*, vol. 1, no. 4, pp. 262–269, 2019.

[47] T. Wu, A. Păun, Z. Zhang, and L. Pan, "Spiking neural P systems with polarizations," *IEEE Transactions on Neural Networks & Learning Systems*, vol. 29, no. 8, pp. 3349–3360, 2018.

[48] T. Song, L. Pan, and G. Păun, "Spiking neural P systems with rules on synapses," *Theoretical Computer Science*, vol. 529, pp. 82–95, 2014.

[49] G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*, Springer, Berlin, Germany, 1997.

[50] L. Pan, G. Păun, T. Wu, and Z. Zhang, "Four recent research topics on numerical and spiking neural P systems," *Romanian Journal of Information Science and Technology*, vol. 19, no. 1-2, pp. 5–16, 2016.

[51] M. Cavaliere, O. H. Ibarra, G. Păun, O. Egecioglu, M. Ionescu, and S. Woodworth, "Asynchronous spiking neural P systems," *Theoretical Computer Science*, vol. 410, no. 24-25, pp. 2352–2364, 2009.

[52] T. Song, L. Pan, and G. Păun, "Asynchronous spiking neural P systems with local synchronization," *Information Sciences*, vol. 219, pp. 197–207, 2013.