Hindawi
Computational Intelligence and Neuroscience
Volume 2020, Article ID 8892240, 10 pages
https://doi.org/10.1155/2020/8892240

Research Article

Turing Universality of Weighted Spiking Neural P Systems with Anti-spikes

Qianqian Ren,[1] Xiyu Liu,[1] and Minghe Sun[2]

[1]*Academy of Management Science, Business School, Shandong Normal University, Jinan, China*
[2]*College of Business, The University of Texas at San Antonio, San Antonio, TX, USA*

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

Received 17 April 2020; Revised 19 July 2020; Accepted 28 August 2020; Published 17 September 2020

Academic Editor: Daniele Bibbo

Weighted spiking neural P systems with anti-spikes (AWSN P systems) are proposed by adding anti-spikes to spiking neural P systems with weighted synapses. Anti-spikes behave like spikes of inhibition of communication between neurons. Both spikes and anti-spikes are used in the rule expressions. An illustrative example is given to show the working process of the proposed AWSN P systems. The Turing universality of the proposed P systems as number generating and accepting devices is proved. Finally, a universal AWSN P system having 34 neurons is proved to work as a function computing device by using standard rules, and one having 30 neurons is proved to work as a number generator.

1. Introduction

Membrane computing, introduced by Păun [1], is a branch of nature-inspired computing. It provides a rich computational framework for biomolecular computing. Models of membrane computing are inspired by the structures and functions of living cells. The obtained models are distributed and parallel computing devices, usually called P systems [2]. There are three main classes of P systems: cell-like P systems, tissue-like P systems [3], and neural-like P systems [4]. Neural-like P systems, inspired by the ways of information storage and processing in human brain nervous systems, are systems that combine neurons and membrane computing, among which the most widely known are spiking neural P systems (SN P systems) [5]. A SN P system consists of a group of neurons located at the nodes of a directed graph, and neurons send spikes to adjacent neurons through synapses, i.e., links in the graph. There is only one type of objects, i.e., spikes, in the neurons.

With different biological features and mathematical motivations, many variants of SN P systems have emerged. Some of them made changes on synapses between neurons, such as SN P systems with rules on synapses [6], SN P systems with multiple channels [7], and SN P systems with thresholds [8], while others made changes on the communication rules, such as SN P systems with communication on request [9], SN P systems with polarizations [10], and SN P systems with inhibitory rules [11]. Various new variants of SN P systems are provided in [12, 13]. Recently, some new variants of neural-like P systems have been proposed, which are inspired by SN P systems, such as those reported in [14]. In addition, many publications appeared in the literature on the computational power of SN P systems as function computing devices and the number generating/accepting devices. Păun [18] proved small universality of SN P systems. Pan [19] proved the small universality of SN P systems with communication on request by using 14 neurons, and more details are available in [20, 21].

Since the SNP system was proposed, many scholars have explored its applications. At present, there are many applications of SN P systems, such as skeletonizing image processing [22, 23], optimization problems [24], fault diagnosis [25–27], and working models [28].

Inspired by the spikes of inhibition of communication between neurons, a new type of SN P systems is proposed by adding anti-spikes to SN P systems, which is called spiking neural P systems with anti-spikes (ASN P systems) [29]. In ASN P systems, each neuron contains multiple copies of symbolic object a or $\overline{a}$ and processes information by spiking rules and forgetting rules. The annihilating rule $a\overline{a} \longrightarrow \lambda$ exists in each neuron and is the first to apply, meaning a and $\overline{a}$ cannot coexist in any neuron. Many researchers have proposed different ASN P systems, such as ASN P systems with multiple channels [30], ASN P systems with rules on synapses [31], and asynchronous ASN P systems [32]. The computational power of ASN P systems as number generating and accepting devices, as well as function computing devices, also can be proved [33].

In [34], SN P systems with weighted synapses were proposed. The weights represent the numbers of synapses between connected neurons. Based on the above, a new variant of SN P systems, called the weighted spiking neural P systems with anti-spikes (AWSN P systems),is proposed in this work. In these systems, neurons receive spikes or anti-spikes from their connected neurons and the numbers of spikes or anti-spikes they receive are determined by the weights of the synapses. Only one type of objects, i.e., spikes or antispikes, exists in each neuron with standard rules in SN P systems. These systems use spiking rules with the form of $(E/a^c) \longrightarrow a^p; d$ (called standard rules if $p = 1$ and extended rules otherwise), where E is a regular expression over spikes a and c, and$p$ and d are all positive integers. The meaning of the spiking rules is that c spikes are consumed and p spikes are generated after d time periods. SN P systems also have forgetting rules of the form $a^s \longrightarrow \lambda$, where s is a positive integer. The meaning of the forgetting rules is that s spikes are dissolved or removed from a neuron.

The rest of this article is organized as follows. In Section 2, the basic knowledge of a register machine is given. The definition of AWSN P systems is given, and an example is presented to show their working process in Section 3. By simulating register machines, the computational power of AWSN P systems is proved as natural number generating devices and accepting devices in Section 4. In Section 5, the universality of these systems as function computing devices and number generating devices is obtained by using 34 neurons and 30 neurons, respectively. Remarks and future research directions are given in Section 6.

2. Prerequisites

The universality of systems is proved by simulating a register machine M. A register machine is structured as $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 and l_h are the starting and ending labels, and I is the set of instructions shown below:

(1) l_i: $(\text{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to instruction labels l_j or l_k with nondeterministic choice)

(2) l_i: $(\text{SUB}(r), l_j, l_k)$ (if register r is not empty, then subtract 1 from it and go to l_j; otherwise, go to l_k)

(3) l_h: HALT (the ending instruction)

A register machine has two modes: a generating mode and an accepting mode. A register machine M generates a set of numbers indefinitely, denoted by $N_{\text{gen}}(M)$, and works in the following way in the generating mode. When all the registers start empty, M starts the computational process from the instruction label l_0. When M reaches l_h, the computation ends with the results stored in register 1. If the computation does not stop, the numbers will not be generated. A set of numbers can also be accepted by a register machine, denoted as $N_{\text{acc}}(M)$, in the accepting mode. Only the input neuron is nonempty at the beginning. It then works in a way similar to that in the generating mode. As register machines are universal in the accepting mode, the add instructions can be written as l_i: $(\text{ADD}(r), l_j)$. Register machines can compute any set of Turing computable numbers represented by NRE (see, e.g., [6]).

Generally, a universal register machine is used to compute Turing computable functions for the purpose of analyzing the computing power of system. A universal register machine M_u is proposed by Minsky [35]. If $\varphi_x(y) = M_u(g(x), y)$ satisfies that x and y are natural numbers and g is a recursive function, then M_u is universal, denoted by $M_u = (8, H, l_0, l_h, I)$, including 8 registers and 23 instructions. Compared with register machine M_u' as shown in Figure 1, register machine M_u does not have instructions l_{22} and l_{23}, and the final result is placed in register 0. Since the result is stored in register 0, it cannot contain any SUB instruction. Hence, register 8 is added and used to store the result without any SUB instruction. In general, in order to analyze the universality of the system, i.e., to verify that the system is equivalent to a Turing machine, a universal register machine M_u'as shown in Figure 1 is simulated by a system, denoted by $M_u' = (9, H, l_0, l_h, I)$, consisting of 9 registers and 25 instructions.

3. Weighted Spiking Neural P Systems with Anti-spikes

3.1. Definition. The proposed AWSN P system is described as follows:

$$\prod = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, \text{syn}, \text{in}, \text{out}), \tag{1}$$

where

(1) $O = \{a, \overline{a}\}$ is the set of alphabets, where the symbol a is a spike, and $\overline{a}$ is an anti-spike.

(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons, in the form of $\sigma_i = (n_i, R_i)$ for $1 \leq i \leq m$, where $n_i \geq 0$ is the initial number of spikes stored in σ_i, and R_i is the set of rules used in σ_i in the following form:

   (a) Spiking rules, $(E/b^c) \longrightarrow b'^p; d$, where E is a regular expression over a or $\overline{a}$, $b, b' \in \{a, \overline{a}\}$, $c \geq p \geq 1$, and $d \geq 0$ are the time unit

   (b) Forgetting rules, $b^s \longrightarrow \lambda$, where $b \in \{a, \overline{a}\}$ and$s \geq 1$

(3) $syn \in \{1, \ldots, h\} \times \{1, \ldots, h\} \times W$ represents the synapses, where $W = \{1, \ldots, n\}$ is the set of weights. For any $(i, j, n) \in syn$, $1 \le i$, $j \le h$, $i \ne j$, and $n \in W$.

(4) in and out are the input neuron and output neuron.

In the AWSN P system, each neuron has one or more spiking rules and some of them also have forgetting rules, and either spikes or anti-spikes exist in each neuron. If there are k spikes or anti-spikes in neuron σ_i, $b^k \in L(E)$ and $k \ge c$, the spiking rule $(E/b^c) \longrightarrow b'^p; d$ can be stimulated. If $k = c$, then the spiking rule is called pure, and the rule can be written as$b^c \longrightarrow b'; d$. The spiking rule can be interpreted as follows. If c spikes or anti-spikes are removed from neuron σ_i and the neuron fires, p spikes will be generated after d time periods (as usual in membrane computing, all neurons in a system Π work in parallel with an assumed global clock) and $p \times n$ spikes will be sent to neuron $\sigma_j (i \ne j)$, where $n \in W$. If the spiking rule of neuron σ_i is used in time d for all $d \ge 1$, the neuron will be closed before time $t + d$ and will not receive any spikes or anti-spikes, and then the neuron will open at time $t + d$. If $t = 0$, spikes will be emitted immediately, which means the neuron receives spikes or anti-spikes from the upper neuron without delay.

If the forgetting rules $b^s \longrightarrow \lambda$ in the neurons are used, then the s spikes or anti-spikes are removed from the neurons. Spiking rules and forgetting rules must be applied if the conditions are met, but the choice of rules is nondeterministic if the conditions of multiple rules are met in a neuron. However, the annihilating rule $a\bar{a} \longrightarrow \lambda$ must be applied first in each neuron.

Through these rules, transitions between configurations can occur. Any sequence of transitions starting from the initial configuration is called a computation. A computation will stop when it reaches a configuration where all neurons are open and no rules can be used. To compute the function $f: N^K \longrightarrow N$, k natural numbers $n_1, n_2 \cdots, n_k$ are introduced into the system by reading a binary sequence $z = 10^{n_1}$, $10^{n_2}1, \ldots, 10^{n_k}1$ from the environment. That is to say, the input neuron of Π receives a spike in a step if it corresponds to 1 in z, but it receives nothing if it corresponds to 0. The input neuron received exactly $k + 1$ spikes and will not receive any more spikes after receiving the last spike. The result of the computation is encoded in the distance between two spikes, which means that the computation halts with exactly two spikes as outputs immediately after outputting the second spike. Hence, it generates a spike string of the form $0^b 10^{r-1}1$, for $b \ge 0$ and $r = f(n_1, \ldots, n_k)$. The computation outputs no spike for a nonspecified number of steps from the beginning of the computation until outputting the first spike.

Let $N_{\text{gen}}(\prod)$ and $N_{\text{acc}}(\prod)$ be the sets of numbers generated and accepted by Π, respectively. Let $N_\alpha ASNP_m^n$, with $\alpha \in \{\text{gen}, \text{acc}\}$, denote the family of sets of numbers generated or accepted by an AWSN P system with m neurons and a maximum of n rules in a neuron.

3.2. An Illustrative Example. An example as graphically shown in Figure 2 is given to explain the working process of the AWSN P system. The results of each step are shown in Table 1. A positive number in the table represents the number of spikes in the neuron, and a negative number represents the number of anti-spikes. For example, 2 means there are two spikes, and −2 means there are two anti-spikes.

The system has four neurons as shown in Figure 2. Assume that each of neurons σ_1 and σ_2 has two spikes, and neurons σ_3 and σ_4 are empty with no spikes. Suppose that the rule $(a^2/a) \longrightarrow \bar{a}$ in neuron σ_1 can be used at time t, generating one anti-spike and sending three anti-spikes to neurons σ_2 and σ_3 because the weight of synapses between these neurons is 3. Two anti-spikes together with two spikes disappear immediately because the annihilating rule is applied first, and there is one anti-spike left in neuron σ_2. The rule in σ_2 generates two spikes to be sent to neuron σ_4 and one spike to be sent to neuron σ_1. So the rule in σ_1 can be applied again. Neuron σ_3 receives six anti-spikes from σ_1 by using the rule of neuron σ_1 twice, so that the rule in σ_3 fires. Neuron σ_4 gets three spikes (two from neuron σ_2) and sends one spike to the environment.

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

Figure 1: The universal register machine M'_u.

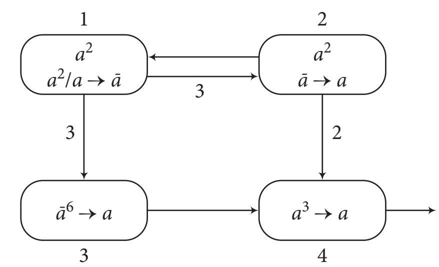

Figure 2: An example of the AWSN P system.

Table 1: The results of the example.

Step	σ_1	σ_2	σ_3	σ_4
t	2	2	0	0
$t+1$	1	−1	−3	0
$t+2$	2	0	−3	2
$t+3$	1	−3	−6	2
$t+4$	1	−3	0	3 (fires)

4. Computational Models

4.1. Generating Mode

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

Proof. A register machine $M = (m, H, l_0, l_h, I)$ is considered. M is simulated by an AWSN P system, including three modules, i.e., modules ADD, SUB, and OUTPUT.

In the simulation process, a register r of M corresponds to neuronσ_r , and the number n contained in register r is the number of spikes contained in neuron σ_r. An instruction l in H corresponds to neuron σ_l. Furthermore, the modules require some other neurons in addition to σ_r and σ_l. The simulation of the ADD and SUB instructions begins at neuron σ_{l_i}. Modules ADD and SUB are simulated by sending spikes to σ_{l_j} and σ_{l_k} as rules in neuron σ_r fire. Neuron σ_r sends a spike to either σ_{l_j} or σ_{l_k}, but the choice is nondeterministic. When a spike arrives at neuron σ_{l_h}, the computation in M stops, and the module OUTPUT begins to send the result stored in register 1 to the environment. At the beginning of the simulation, neuron σ_{l_0} has one spike but other neurons do not have any spikes.

(a) Module ADD (Shown in Figure 3) Assume that an ADD instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$ has to be simulated at time t, one spike is in neuron σ_{l_i}, and the rule $a \longrightarrow a$ can be used. Neuron σ_{l_i} sends one spike a to neurons σ_r, σ_{b_1}, and σ_{b_2}, respectively. The rules $a \longrightarrow \overline{a}$ and $a \longrightarrow a$ in neuron σ_{b_1} are chosen in a nondeterministic way for use at time $t+1$. In this way, there are two cases to consider depending on the choice of the rules in σ_{b_1}. If $a \longrightarrow a$ is chosen, neuron σ_{b_2} sends a spike to neuron σ_{l_k}. Thus, σ_{l_k} will generate one spike by using its rule. If $a \longrightarrow \overline{a}$ is chosen, neuron σ_{b_1} sends an anti-spike to neurons σ_{l_i} and σ_{b_2}, respectively. Thus σ_{l_j} will fire and generate one spike by using its rule. The rule in neuron σ_{b_2} cannot be used because of the annihilating rule, so that σ_{l_k} is empty. After one spike is added to σ_r, the register r adds 1 and the instruction l_j or l_k is activated. Therefore, the ADD instruction can be simulated correctly by the module ADD.

(b) Module SUB (Shown in Figure 4) Suppose that neuron σ_{l_i} has one spike. After the rule $a \longrightarrow \overline{a}$ is enabled at time t, each of the neurons σ_{l_j} and σ_{l_k} receives two anti-spikes $\overline{a}$, and σ_r receives one anti-spike. The rest of the computation can be divided into two cases according to the number of spikes contained in σ_r.

(1) *Neuronσ_r has at least one spike.* Neuron σ_r receives one anti-spike from neuron σ_{l_i}, but anti-spike will disappear immediately by annihilating one spike in σ_r. Therefore, the rule $\overline{a} \longrightarrow a$ in neuron σ_r is not used at time $t+1$. At the same time, neuron σ_{c_1} opens to get one anti-spike from σ_{l_i}, and then the rule in σ_{c_1} fires and generates one spike but sends three spikes to neurons σ_{l_j} and two spikes to σ_{l_k}. The two spikes are annihilated with two anti-spikes from σ_{l_i} and one spike is left in neuron σ_{l_j}. Simultaneously, the same happens in neuron σ_{l_k}, i.e., the two spikes are annihilated immediately and there is no spike left in σ_{l_k}.

(2) *Neuronσ_r has no spike.* Neuron σ_r gets one anti-spike from σ_{l_i} and its rule can be applied at time $t+1$. Simultaneously, neuron σ_{c_1} gets one anti-spike from σ_{l_i}. Hence, one spike from σ_r is annihilated in the next time. The rule in σ_r cannot be used because σ_r does not have any anti-spikes. At the same time, neuron σ_{l_j} receives five spikes, among which two spikes are used to annihilate the two anti-spikes received from neuron σ_{l_i}; thus the rule $a^2 \longrightarrow \lambda$ in σ_{l_j} can be applied. Neuron σ_{l_k} receives one spike that annihilates one anti-spike$\overline{a}$ received from neuron σ_{l_i}, and then the rule $\overline{a} \longrightarrow a$ in σ_{l_k} is enabled to generate one spike a.

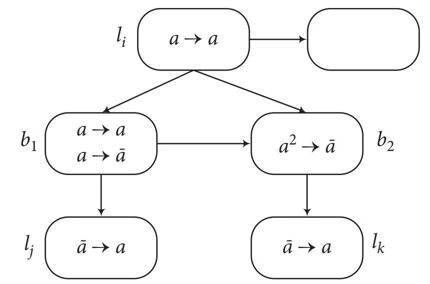

Figure 3: Module ADD: stimulating the ADD instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$.

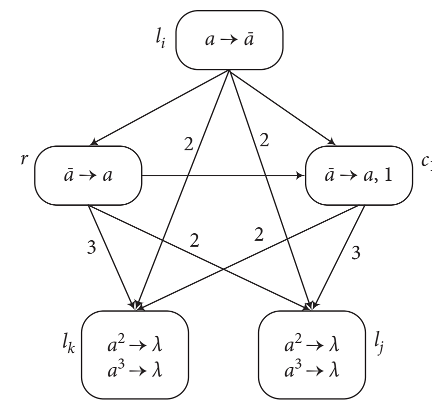

Figure 4: Module SUB: simulating the SUB instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$.

Therefore, the SUB instruction can be simulated correctly by module SUB.

(c) Module OUTPUT (Shown in Figure 5) Assume that σ_{l_h} of system $\prod$ accumulated one spike at time t, and neuron σ_1 has n spikes for the number n being stored in register 1 of M. When the rule in $\sigma_{l'_h}$ is fired at time t, neuron $\sigma_{l'_h}$ sends one spike to σ_1. At this moment, σ_1 has an odd number of spikes and its rule fires. At time $t+1$, σ_1 sends one spike to σ_{out} and σ_{b_1}, respectively. Thus, neuron σ_{out} has one spike, which is an odd number. At time $t+2$, neuron σ_{out} fires,

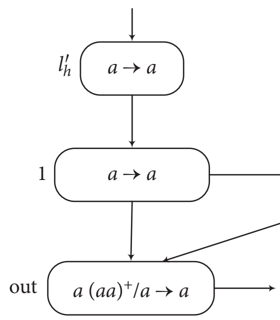

FIGURE 5: Module OUTPUT.

sending one spike to the environment. At the same time, the rules in σ_1 and σ_{b_1} are used, and both send one a to σ_{out}. After $n-1$ steps, until neuron σ_1 has no spike, the number of spikes in σ_{out} is even. At the same time, the use of the rule in σ_1 is stopped, and neuron σ_{b_1} has one spike. Neuron σ_{out} will receive one spike at time $t+n+2$, and then the number of spikes is odd. Neuron σ_{out} fires a second time. Therefore, the number computed by the AWSN P system is the difference between the first two steps when the neuron σ_{out} fires; that is, $(t+n+2)-(t+2)=n$. The module OUTPUT can be simulated correctly.

4.2. Accepting Mode

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

Proof. The proof of this theorem is similar to that of Theorem 1. A register machine $M=(m,H,l_0,l_h,I)$, consisting of three modules, ADD, SUB, and INPUT, is considered. Module SUB is shown in Figure 4.

(1) Module ADD (Shown in Figure 6) Assume that an ADD instruction l_i: $(\text{ADD}(r), l_j)$ has to be simulated at time t. Suppose that one spike is in neuron σ_{l_i}; then the rule $a \longrightarrow a$ can be used. Thus, neuron σ_{l_i} sends one spike to neurons σ_r and σ_{l_j}. In this way, the number of spikes in σ_r increases by 1 and the instruction l_j is activated. Hence, the ADD instruction can be simulated correctly by this module.

(2) Module INPUT (Shown in Figure 7) Module INPUT shown in Figure 7 works as follows. The function of module INPUT is to read the spike train $10^{n-1}1$ and compute the number n in the time between receiving two spikes. When neuron σ_{in} receives the first spike at time$t$ and then neurons σ_{d_1},σ_{d_2}, and σ_{d_3} receive one spike each, the rule in $\sigma_{d_2}$and σ_{d_3} can be applied at time$t+1$. At time$t+2$, neuron σ_1 gets one spike, and, at the same time, neuron σ_{d_3} gets one spike from σ_{d_2} and neuron σ_{d_2} receives one a from σ_{d_3}. Therefore, in the next $n-1$time periods, the rules in neurons σ_{d_2} and σ_{d_3} can continued to be used.

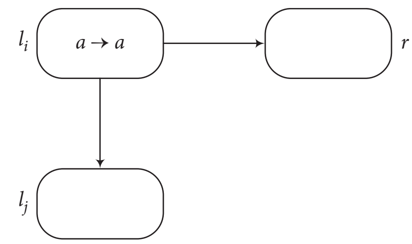

FIGURE 6: Module ADD: simulating the ADD instruction l_i: $(\text{ADD}(r), l_j)$.

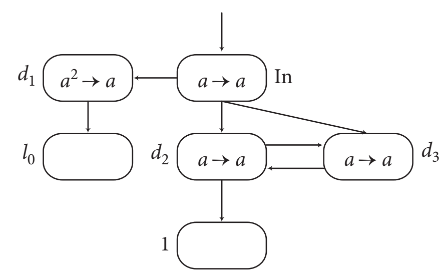

FIGURE 7: Module INPUT.

During this period, σ_1 gets $n-1$ spikes. When neuron σ_{in} receives the second spike at step $t+n$, each of neurons σ_{d_2} and σ_{d_3} receives one spike at step $t+n+1$ and they both have two spikes. In this way, neurons σ_{d_2} and σ_{d_3} cannot fire to send any spikes to neuron σ_1. In the whole process, neuron σ_1 receives $(n-1)+1=n$ spikes, i.e., the number $n$is stored in register 1.

From the descriptions above about the three modules, it is clear that the register machine M can correctly simulate the system. The proof is complete.

5. A Small Universal AWSN P System

5.1. The Universality as Function Computing Devices

Theorem 3. *There is a universal AWSN P system having 34 neurons which can be used to perform function computing.*

Proof. A general framework of a system $\prod_u'$ used to simulate a universal register machine M_u' is shown in Figure 8, which is a universal AWSN P system. Π' consists of 8 modules: ADD, SUB, ADD-ADD, SUB-ADD-1, SUB-ADD-2, SUB-SUB, INPUT, and OUTPUT. The modules SUB, OUTPUT, and ADD are the same as those in Figures 4–6, respectively. The module INPUT is shown in Figure 9.

Module INPUT works as follows: when neuron σ_{in} gets a spike from the environment, the rule $a \longrightarrow a$ fires and one spike is sent to neurons σ_{c_1}, σ_{c_3}, and σ_{c_4}, and two spikes are sent to neuron σ_{c_2}. Then, the rule in neuron σ_{c_1} sends one spike to both σ_{c_2} and σ_1. At the same time, neuron σ_{c_2} fires and then sends one spike to σ_{c_1} and two spikes to σ_{c_3}.

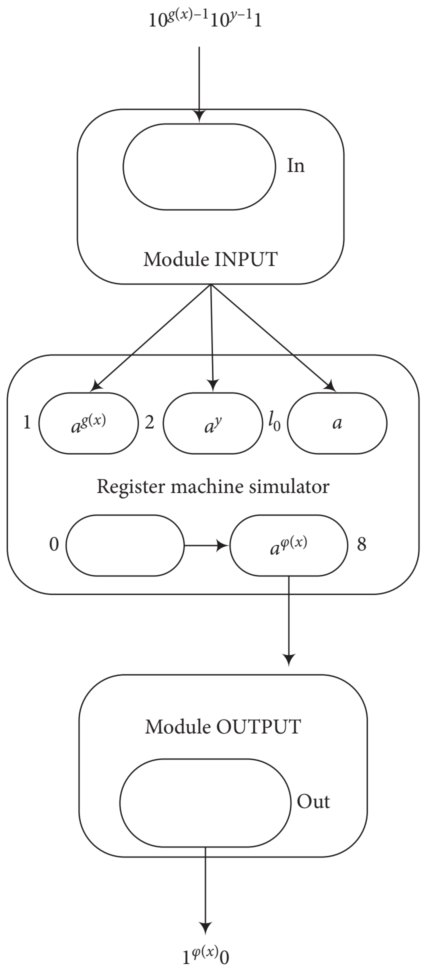

Figure 8: General framework of the universal AWSN P system.

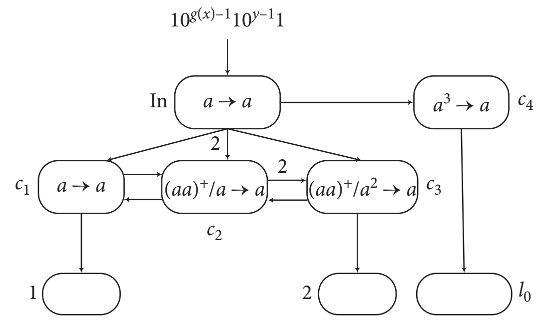

Figure 9: Module INPUT.

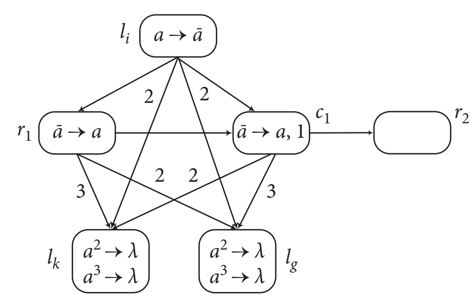

Figure 10: Module SUB-ADD-1: the sequence of the ADD and SUB instructions l_j: $(\mathrm{ADD}(r_2), l_g)$ and l_i: $(\mathrm{SUB}(r_1), l_j, l_k)$.

Up to this point, three spikes were sent to neuron σ_{c_3}. Therefore, before neuron σ_{in} receives more spikes from the environment, neurons σ_{c_1} and σ_{c_2} have received one spike from each other in each time period and neuron σ_1 has received $g(x)$ spikes.

When σ_{in} receives the second spike, each of the neurons σ_{c_1}, σ_{c_3}, and σ_{c_4} can get one spike and σ_{c_2} gets two spikes. Neuron σ_{c_2} has four spikes at this moment, and its rule can be used to send two spikes to neuron σ_{c_3}. Neuron σ_{c_3} then has six spikes, so that the rule in σ_{c_3} is used to produce one spike and send it to σ_{c_2}. In this way, neurons σ_{c_2} and σ_{c_3} receive one spike from each other in each step before σ_{in} receives the third spike from the environment. Neuron σ_2 has y spikes at the end. When neuron σ_{in} receives the third spike, each of the neurons σ_{c_1}, σ_{c_3}, and σ_{c_4} gets one spike, while σ_{c_2} receives two spikes. As a result, neuron σ_{c_3} has an odd number of spikes and the rule cannot be applied. At present, neuron σ_{c_4} has three spikes, and the rule $a^3 \longrightarrow a$ in neuron σ_{c_4} fires, which generates one spike and sends it to σ_{l_0}. In this way, it can simulate the instruction l_0 in the next step.

As with the proof of Theorems 1 and 2, the system uses the following numbers of neurons:

9 neurons for 9 registers

25 neurons for 25 labels

5 neurons for the module INPUT

1 neuron in each SUB instructions and 14 in total

2 neurons for the module OUTPUT

Therefore, totally 55 neurons are used.

The numbers of neurons can be decreased by exploring some relationships between some instructions of register machine M_u'. The following modules are given to reduce the number of neurons in the computation process.

The SUB-ADD instructions can be divided into two cases, depending on the number of spikes placed in register r_1 (the register involved in the SUB instruction). Modules SUB-ADD-1 and SUB-ADD-2 shown in Figures 10 and 11 can simulate the SUB and ADD instructions sequentially. The working process of module SUB-ADD-1 is similar to that of module SUB. When the rule in neuron σ_{l_i} is used and σ_{r_1} contains at least one spike, neuron σ_{r_1} cannot fire. Neuron σ_{c_1} fires by receiving one $\overline{a}$ and then sends one spike to σ_{r_2}. At the end of the computation, neuron σ_{l_g} has one spike, neuron σ_{r_2} has one spike, and neuron σ_{l_k} is empty. When σ_{r_1} is empty, neurons σ_{r_2} and σ_{l_g} are also empty and neuron σ_{l_k} contains one spike. Thus, each pair of SUB-ADD-1 instructions l_i: $(\mathrm{SUB}(r_1), l_j, l_k)$ and l_j: $(\mathrm{ADD}(r_2), l_g)$ can share a common neuron when $r_1 \neq r_2$, and there are totally 6 pairs in M_u':

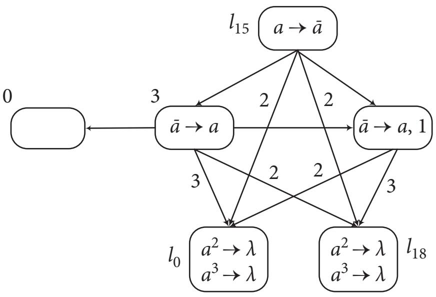

FIGURE 11: Module SUB-ADD-2: the sequence of the ADD and SUB instructions l_{20}: $(\text{ADD}(0), l_0)$ and l_{15}: $(\text{SUB}(3), l_{18}, l_{20})$.

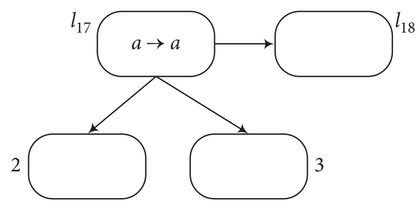

FIGURE 12: Module ADD-ADD: the sequence of ADD and ADD instructions l_{17}: $(\text{ADD}(2), l_{21})$ and l_{21}: $(\text{ADD}(3), l_{18})$.

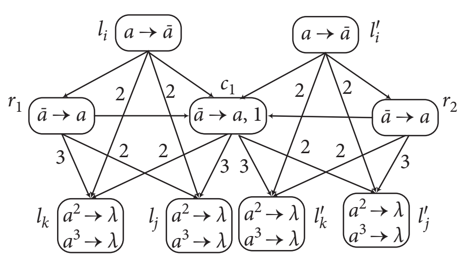

FIGURE 13: Module SUB-SUB with $r_1 \neq r_2$.

$$
\begin{aligned}
l_0&: \left(\text{SUB}(1), l_1, l_2\right),\\
l_1&: \left(ADD(7), l_0\right),\\
l_4&: \left(\text{SUB}(6), l_5, l_3\right),\\
l_5&: \left(ADD(5), l_6\right),\\
l_6&: \left(\text{SUB}(7), l_7, l_8\right),\\
l_7&: \left(ADD(6), l_4\right),\\
l_8&: \left(\text{SUB}(6), l_9, l_0\right),\\
l_9&: \left(ADD(6), l_{10}\right),\\
l_{14}&: \left(\text{SUB}(5), l_{16}, l_{17}\right),\\
l_{16}&: \left(ADD(4), l_{11}\right),\\
l_{22}&: \left(\text{SUB}(0), l_{23}, l_h'\right),\\
l_{23}&: \left(ADD(0), l_0\right).
\end{aligned}
\tag{2}
$$

By using this module, 6 neurons can be saved. In the same way, the module shown in Figure 10 can simulate the two instructions l_{15} and l_{20}. Neuron $\sigma_{l_{20}}$ can be saved.

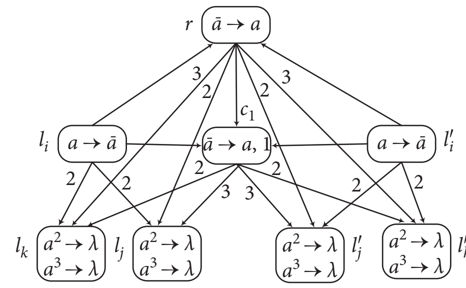

FIGURE 14: Module SUB-SUB with $r_1 = r_2$.

The module ADD-ADD shown in Figure 12 can simulate instructions l_{17} and l_{21}. In this way, one neuron can be saved.

The SUB instructions share a common neuron when the labels of their registers are different, as shown in Figure 13. Assume that the simulation of the SUB instruction l_i: $(\text{SUB}(r_1), l_j, l_k)$ starts at time t. When neuron σ_{l_i} gets a spike, the rule $a \longrightarrow \bar{a}$ fires and sends one anti-spike to σ_{r_1} and two anti-spikes to σ_{l_j} and σ_{l_k}, respectively, at time $t+1$. Neuron σ_{c_1} receives an anti-spike at time $t+2$. Neurons σ_{r_1}, σ_{l_j}, σ_{l_k}, and σ_{c_1} work in the same way as those in module SUB shown in Figure 4. Neuron σ_{c_1} will send three spikes to $\sigma_{l_k'}$ and two spikes to $\sigma_{l_j'}$, where forgetting rules will be applied. Thus, the instruction l_i: $(\text{SUB}(r_1), l_j, l_k)$ is correctly simulated by this module. The process when starting with instruction l_i' is similar to that described above.

Two SUB modules dealing with the same register, as shown in Figure 14, can also be proved to work correctly in a similar way. Assume that the instruction l_i: $(\text{SUB}(r_1), l_j, l_k)$ is simulated and one spike is contained in neuron σ_{l_i}. The process is divided into two cases according to the number of spikes in neuron σ_r. When σ_r has at least one spike, the working process of the system is similar to that of module SUB. When σ_r is empty, the rule in neuron σ_{c_1} cannot be used. Neurons σ_{l_j}, $\sigma_{l_k'}$, and $\sigma_{l_j'}$ are all empty but neuron σ_{l_k} contains one spike. All SUB instructions can be simulated correctly by the module. Therefore, all SUB modules can share a common neuron.

From the above description about the numbers of neurons saved, the system uses the following:

9 neurons for 9 registers

17 neurons for 17 labels

5 neurons for the module INPUT

1 neuron for all the 14 SUB instructions

2 neurons for the module OUTPUT

A total of 21 neurons can be saved and the number of neurons in this system can be decreased from 55 to 34. The proof is complete. □

5.2. The Small Universality as Number Generator. A small universal AWSN P system as a number generator is considered. The process of simulating universal number generators is similar to that of simulating general function computing devices, but the difference between them lies in the module INPUT. The system starts with the spike train

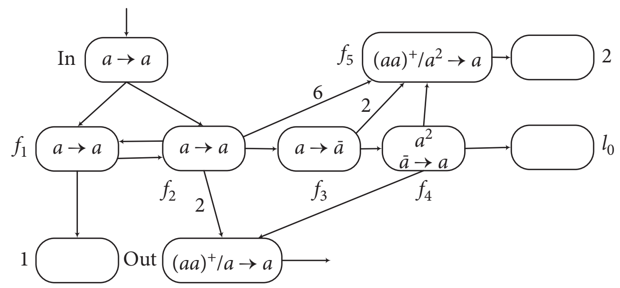

Figure 15: Module INPUT-OUTPUT.

Table 2: The computation process of the module INPUT-OUTPUT.

Step	σ_{in}	σ_{f_1}	σ_{f_2}	σ_{f_3}	σ_{f_4}	σ_{f_5}	σ_1	σ_2	σ_{out}	σ_{l_0}
t	1	0	0	0	2	0	0	0	0	0
$t+1$	0	1	1	0	2	0	0	0	0	0
$t+2$	1	1	1	1	2	6	1	0	2 (fire)	0
$t+3$	0	2	2	1	1	8	2	1	3	0
$t+4$	0	2	2	0	0	4	2	2	3	0
$t+5$	0	2	2	0	−1	2	2	3	3	0
$t+6$	0	2	2	0	0	1	2	4	4 (fire)	1

$10^{g(x)-1}1$ from environment and ends with neuron σ_1 receiving$g(x)$ spikes. This system is then loaded with an arbitrary number k, and neuron σ_2 receives k spikes. The number k is also the output at the same time as the output spike train $10^{g(x)-1}1$, with $g(x)$ in register 1 and k in register 2. Since the output module is not required, that is to say, register 8 is not required, the register machine M_u is simulated. If the computation in M_u halts, the computation can also halt.

Furthermore, module INPUT and module OUTPUT can be combined. The module INPUT-OUTPUT is shown in Figure 15, and an example is used to prove its feasibility. The label l_h' can also be saved because of module INPUT-OUTPUT. The string 101 is used in module INPUT-OUTPUT, where $g(x) = 2$ and $k = 4$. The computation follows the above working processes of the modules. The results of each step are shown in Table 2.

Assume that σ_{in} has one spike at time$t$, and neuron σ_{f_4} has two spikes. At time$t + 1$, σ_{f_1} and σ_{f_2} receive one spike, respectively. From the structure shown in Figure 15, neurons σ_{f_1} and σ_{f_2} receive one spike from each other at each step until σ_{f_1} and σ_{f_2}stop firing. Then σ_{in} receives the second spike. Each of neurons σ_1 and σ_{f_3} receives one spike, σ_{f_5} receives six spikes, and σ_{out} receives two spikes, so that neurons σ_{f_5} and σ_{out} can fire. At time$t + 3$, both σ_{f_1} and σ_{f_2} have two spikes, but they cannot fire again. σ_{f_5} receives six spikes from σ_{f_2}, but σ_{f_5} also receives two anti-spikes from σ_{f_3}, plus four spikes existing in σ_{f_5}, so that neuron σ_{f_5} has eight spikes. In addition, neuron σ_{out} receives two spikes again, so that there are three spikes contained. Neuron σ_{f_4} only has one spike because the received anti-spike annihilates one spike. At time$t + 4$, the neuron σ_{f_4} is empty after receiving an anti-spike. σ_{f_5} receives two anti-spikes, so that there are four spikes contained in neuron σ_{f_5}, the number of spikes is even, and its rule can fire. At the next step, σ_{f_4} receives one anti-spike and fires. Neuron σ_{f_5} consumes two spikes and still can fire. At time$t + 6$, neurons σ_{f_5} and σ_{out} receive one spike from σ_{f_4}, respectively. So, there are 4 spikes in σ_{out}, meeting the required conditions for firing. Neuron σ_{l_0} also gets one spike.

The string is read through neuron σ_{in}, and g(x) spikes are stored in register 1 when the calculation stops. At the same time , the output number ($t + 6 - t - 2 = 4$) is the same as the number stored in register 2. Neuron σ_{l_0} activates and starts simulating the register machine by simulating modules ADD and SUB. Therefore, through this process, the module INPUT-OUTPUT can be simulated correctly.

Therefore, this system contains the following:

- 8 neurons for the 8 registers
- 14 neurons for the 14 labels (l_h is saved; 8 neurons are saved by modules SUB-ADD and ADD-ADD)
- 1 neuron for 13 SUB instructions
- 7 neurons in the module INPUT-OUTPUT

There is a universal AWSN P system having 30 neurons that can be used to perform number generating.

6. Conclusions

In this work, a variant of the SN P systems, called the AWSN P systems,is proposed. Because of the use of anti-spikes, the proposed systems are more biologically significant thanSN P systems, with inhibitory spikes in the communication between neurons. An example is used to illustrate the working process of this system. The computational universality is then proved in the case of generating mode and accepting mode, respectively. Finally, the Turing universality of AWSN P systems is proved. The function computing device can be realized by using 34 neurons. Compared with the small universal SN P system using anti-spikes introduced by Song [17], the AWSN P system uses 13 fewer neurons. Compared with the SN P systems with weighted synapses introduced by Pan [34], the AWSN P system uses 4 fewer neurons. The small universality of the ASN P system as number generator is investigated with 30 neurons. Compared with Pan's work [34], the proposed system uses 6 fewer neurons.

The computational universality is proved for AWSN P systems with standard rules. There are three types of spiking rules, $a \longrightarrow \bar{a}$, $a \longrightarrow a$, and $\bar{a} \longrightarrow a$, used that are time dependent, and there is one type of forgetting rules, $a^c \longrightarrow \lambda$. There are several future research directions. One direction is to investigate whether the computational power will remain the same if only one or two types of spiking rules are used or if the forgetting rules are not used and to investigate whether AWSN P systems can perform better or the same if the spiking rules are not time-dependent. These open problems certainly need further studies. Another future research direction is the application of the proposed systems. There have been studies, such as using SN P systems with learning function for letter recognitions [36]. If the learning function was introduced in AWSN P systems, it may perform better in letter recognitions. Because the use of anti-spikes improves the ability of AWSN P systems to represent and process information, it may solve more practical problems, which still require further research.

Data Availability

No datasets were used in this article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

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

References

[1] G. Păun, "Computing with membranes," *Journal of Computer & System Sciences*, vol. 61, pp. 108–143, 2000.

[2] G. Păun, G. Rozenberg, and A. Salomaa, *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, UK, 2010.

[3] C. Martin-Vide, J. Pazos, G. Păun, and A. Rodríguez-Patón, "Tissue P systems," *Theoretical Computer Science*, vol. 296, pp. 295–326, 2003.

[4] G. Păun, *Membrane Computing: An Introduction*, Springer-Verlag, Berlin, Germany, 2012.

[5] M. Ionescu, G. Păun, and "T. Yokomori, "Spiking neural P systems," *Fundamenta Informaticae*, vol. 71, pp. 279–308, 2006.

[6] T. Song, L. Pan, and G. Păun, "Spiking neural P systems with rules on synapses," *Theoretical Computer Science*, vol. 529, pp. 888–895, 2014.

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

[8] 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.

[9] L. Pan, G. Păun, G. Zhang, and F. Neri, "Spiking neural P systems with communication on request," *International Journal of Neural Systems*, vol. 27, pp. 1–17, 2017.

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

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

[12] A. Alhazov, R. Freund, S. Ivanov, M. Oswald, and S. Verlan, "Extended spiking neural P systems with white hole rules and their red-green variants," *Natural Computing*, vol. 17, no. 2, pp. 297–310, 2017.

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

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

[15] 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.

[16] 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.

[17] T. Song, Y. Jiang, X. Shi, and X. Zeng, "Small universal spiking neural P systems with anti-spikes," *Journal of Computational and Theoretical Nanoscience*, vol. 10, no. 4, pp. 999–1006, 2013.

[18] A. Pǎun and G. Pǎun, "Small universal spiking neural P systems," *BioSystems*, vol. 90, pp. 48–60, 2007.

[19] T. Pan, X. Shi, Z. Zhang, and F. Xu, "A small universal spiking neural P system with communication on request," *Neurocomputing*, vol. 275, pp. 1622–1628, 2017.

[20] X. Zhang, X. Zeng, and L. Pan, "Smaller universal spiking neural P systems," *Fundamenta Informaticae*, vol. 87, pp. 117–136, 2008.

[21] T. Wu, F. Bîlbîe, A. Păun, L. Pan, and F. Neri, "Simplified and yet turing universal spiking neural P systems with communication on request," *International Journal of Neural Systems*, vol. 28, pp. 1–19, 2018.

[22] 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.

[23] T. Song, S. Pang, S. Hao, A. Rodríguez-Patón, and P. Zheng, "A parallel image skeletonizing method using spiking neural P systems with weights," *Neural Processing Letters*, vol. 115, 2018.

[24] 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, 2014.

[25] Y. Yahya, A. Qian, and A. Yahya, "Power transformer fault diagnosis using fuzzy reasoning spiking neural P systems," *Journal of Intelligent Learning Systems and Applications*, vol. 8, no. 4, pp. 77–91, 2016.

[26] T. Wang, J. Zhao, G. Zhang, Z. He, and J. Wang, "Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems," *IEEE Transactions on Power Systems*, vol. 30, pp. 1182–1194, 2014.

[27] T. Wang, X. Wei, J. Wang et al., "A weighted corrective fuzzy reasoning spiking neural P system for fault diagnosis in power systems with variable topologies," *Engineering Applications of Artificial Intelligence*, vol. 92, 2020.

[28] T. Song, X. Zeng, P. Zheng, M. Jiang, and A. Rodríguez-Patòn, "A parallel workflow pattern modeling using spiking neural P systems with colored spikes," *IEEE Transactions on Nanobioscience*, vol. 17, no. 4, pp. 474–484, 2018.

[29] L. Pan and G. Păun, "Spiking neural P systems with anti-spikes," *International Journal of Computers Communications & Control*, vol. 4, no. 3, pp. 273–282, 2009.

[30] X. Song, J. Wang, H. Peng et al., "Spiking neural P systems with multiple channels and anti-spikes," *Biosystems*, vol. 169-170, no. 170, pp. 13–19, 2018.

[31] T. Wu, Y. Wang, S. Jiang, Y. Su, and X. Shi, "Spiking neural P systems with rules on synapses and anti-spikes," *Theoretical Computer Science*, vol. 724, pp. 13–27, 2018.

[32] T. Song, X. Liu, and X. Zeng, "Asynchronous spiking neural P systems with anti-spikes," *Kluwer Academic Publishers*, vol. 42, pp. 633–647, 2015.

[33] V. P. Metta and A. Kelemenová, "Universality of spiking neural P systems with anti-spikes," in *Proceedings of the International conference on theory & application of models and computation*, pp. 352–365, New York, NY. USA, 2014.

[34] L. Pan, X. Zeng, X. Zhang, and Y. Jiang, "Spiking neural P systems with weighted synapses," *Neural Processing Letters*, vol. 35, no. 1, pp. 13–27, 2012.

[35] M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, New Jersey, NJ, USA, 1967.

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