Research Article

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

Qianqian Ren,1 Xiyu Liu,1 and Minghe Sun2

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

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

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

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 \(a\overline{a} \rightarrow \lambda\) 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 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 (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 \(E/a^p \rightarrow a^q; d\) (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 \(a^s \rightarrow \lambda\), 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 \(M = (m, H, l_0, l_n, I)\), where \(m\) is the number of registers, \(H\) is the set of instruction labels, \(l_0\) and \(l_n\) are the starting and ending labels, and \(I\) is the set of instructions shown below:

1. \(l_i: (ADD(r), l_j, l_k)\) (add 1 to register \(r\) and then go to instruction labels \(l_j\) or \(l_k\) with nondeterministic choice)
2. \(l_i: (SUB(r), l_j, l_k)\) (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 \(N_{gen}(M)\), 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 \(l_\text{acc}\), 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 \(N_{acc}(M)\), 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 \(l'_i: (ADD(r), l_j)\). 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 \(\varphi_s(y) = M_u(g(x), y)\) satisfies that \(x\) and \(y\) are natural numbers and \(g\) is a recursive function, then \(M_u\) is universal, denoted by \(M_u = (8, H, l_0, l_n, I)\), 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 \(l_3\), 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 \(M'_u = (9, H, l_0, l_n, I)\), 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:

\[
\prod = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, \text{syn}, \text{in}, \text{out}),
\]

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 \(\sigma_i\), and \(R_i\) is the set of rules used in \(\sigma_i\) in the following form:
   
   a. Spiking rules, \((E/b^p) \rightarrow b^{p'}; d\), 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, \(b^d \rightarrow \lambda\), where \(b \in \{a, \overline{a}\}\) and \(d \geq 1\)
3. syn ∈ \{1, \ldots, h\} × \{1, \ldots, h\} × W \) represents the synapses, where \( W = \{1, \ldots, n\} \) is the set of weights.

For any \((i, j, n) ∈ \text{syn}, 1 ≤ i, j ≤ h, i \neq j, \) and \( n ∈ W \).

(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 \( σ_i \), \( b^k ∈ L(E) \) and \( \max(\sigma_i) ≥ c \), the spiking rule \((E/b^k) \rightarrow b^p; d \) can be stimulated. If \( k = c \), the spiking rule is called pure, and the rule can be written as \( b^c \rightarrow b^p; d \). 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 \( \Pi \) work in parallel with an assumed global clock) and \( p × n \) spikes will be sent to neuron \( σ_j \) (\( i \neq j \)), where \( n ∈ W \).

If the spiking rule of neuron \( σ_i \), is used in time \( d \) for all \( d \geq 1 \), 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 \( b^p \rightarrow λ \) 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 \( aλ \rightarrow λ \) 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 \( f: N^k → N \), \( k \) natural numbers \( n_1, n_2, \ldots, n_k \) are introduced into the system by reading a binary sequence \( z = 10^n, 10^n1, \ldots, 10^n1 \) 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 \( σ_1 \) 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 \( 0^b10^{r−1}1 \), for \( b ≥ 0 \) and \( r = f(n_1, \ldots, n_k) \). The computation outputs no spikes for a nonspecified number of steps from the beginning of the computation until outputting the first spike.

Let \( N_{\text{gen}}(\Pi) \) and \( N_{\text{acc}}(\Pi) \) be the sets of numbers generated and accepted by \( Π \), respectively. Let \( N_{\text{ASNP}}^m \) with \( a ∈ \{\text{gen, acc}\} \), denote the family of sets of numbers generated or accepted by an AWNS 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 AWNS 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 \( (a^2/a) → π \) 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 \( σ_1 \). The rule in \( σ_2 \) generates two spikes to be sent to neuron \( σ_3 \) 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 \( σ_2 \) fires. Neuron \( σ_1 \) gets three spikes (two from neuron \( σ_2 \) and one anti-spike from \( σ_3 \)) and sends one spike to the environment.

![Figure 1: The universal register machine \( M'_u \).](image1)

![Figure 2: An example of the AWNS P system.](image2)

<table>
<thead>
<tr>
<th>Step</th>
<th>( σ_1 )</th>
<th>( σ_2 )</th>
<th>( σ_3 )</th>
<th>( σ_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t + 1 )</td>
<td>1</td>
<td>−1</td>
<td>−3</td>
<td>0</td>
</tr>
<tr>
<td>( t + 2 )</td>
<td>2</td>
<td>0</td>
<td>−3</td>
<td>2</td>
</tr>
<tr>
<td>( t + 3 )</td>
<td>1</td>
<td>−3</td>
<td>−6</td>
<td>2</td>
</tr>
<tr>
<td>( t + 4 )</td>
<td>1</td>
<td>−3</td>
<td>0</td>
<td>3 (fires)</td>
</tr>
</tbody>
</table>

Table 1: The results of the example.
4. Computational Models

4.1. Generating Mode

Theorem 1. \( N_{\text{gen ASNP}}^2 = \text{NRE} \).

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 neuron \( \sigma_r \), and the number \( n \) contained in register \( r \) is the number of spikes contained in neuron \( \sigma_r \). An instruction \( I \) in \( H \) corresponds to neuron \( \sigma_I \). Furthermore, the modules require some other neurons in addition to \( \sigma_r \) and \( \sigma_I \). The simulation of the ADD and SUB instructions begins at neuron \( \sigma_I \). Modules ADD and SUB are simulated by sending spikes to \( \sigma_1 \) and \( \sigma_2 \) as rules in neuron \( \sigma_f \). Neuron \( \sigma_r \) sends a spike to either \( \sigma_1 \) or \( \sigma_2 \), but the choice is nondeterministic. When a spike arrives at neuron \( \sigma_r \), the computation in \( M \) stops, and the module OUTPUT begins to send the result stored in register \( I \) to the environment. At the beginning of the simulation, neuron \( \sigma_1 \) has one spike but other neurons do not have any spikes.

(a) Module ADD (Shown in Figure 3) Assume that an ADD instruction \( l_1 \) : (ADD(r), \( I_1, I_k \) ) has to be simulated at time \( t \), one spike is in neuron \( \sigma_1 \), and the rule \( a \rightarrow a \) can be used. Neuron \( \sigma_1 \) sends one spike \( a \) to neurons \( \sigma_2, \sigma_3, \sigma_b, \) and \( \sigma_b \), respectively. The rules \( a \rightarrow a \) and \( a \rightarrow a \) in neuron \( \sigma_3 \) 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 \( \sigma_3 \). If \( a \rightarrow a \) is chosen, neuron \( \sigma_2 \) sends a spike to neuron \( \sigma_3 \). Thus, \( \sigma_3 \) will generate one spike by using its rule. If \( a \rightarrow a \) is chosen, neuron \( \sigma_3 \) sends an anti-spike to neurons \( \sigma_1 \) and \( \sigma_b \), respectively. Thus, \( \sigma_3 \) will fire and generate one spike by using its rule. The rule in neuron \( \sigma_3 \) cannot be used because of the annihilating rule, so that \( \sigma_3 \) is empty. After one spike is added to \( \sigma_r \), the register \( r \) adds 1 and the instruction \( I_1 \) or \( I_k \) is activated. Therefore, the ADD instruction can be simulated correctly by the module ADD.

(b) Module SUB (Shown in Figure 4) Suppose that neuron \( \sigma_1 \) has one spike. After the rule \( a \rightarrow a \) is enabled at time \( t \), each of the neurons \( \sigma_1 \) and \( \sigma_2 \) receives two anti-spikes \( \lambda \), and \( \sigma_r \) receives one anti-spike. The rest of the computation can be divided into two cases according to the number of spikes contained in \( \sigma_r \).

(1) Neuron \( \sigma_1 \) has at least one spike. Neuron \( \sigma_1 \) receives one anti-spike from neuron \( \sigma_2 \), but anti-spike will disappear immediately by annihilating one spike in \( \sigma_1 \). Therefore, the rule \( \lambda \rightarrow a \) in neuron \( \sigma_r \) is not used at time \( t + 1 \). At the same time, neuron \( \sigma_1 \) opens to get one anti-spike from \( \sigma_1 \), and then the rule in \( \sigma_1 \) fires and generates one spike but sends three spikes to neurons \( \sigma_1 \) and two spikes to \( \sigma_2 \). The two spikes are annihilated with two anti-spikes from \( \sigma_1 \) and one spike is left in neuron \( \sigma_1 \). Simultaneously, the same happens in neuron \( \sigma_2 \), i.e., the two spikes are annihilated immediately and there is no spike left in \( \sigma_2 \).

(2) Neuron \( \sigma_1 \) has no spike. Neuron \( \sigma_r \) gets one anti-spike from \( \sigma_1 \) and its rule can be applied at time \( t + 1 \). Simultaneously, neuron \( \sigma_2 \) gets one anti-spike from \( \sigma_1 \). Hence, one spike from \( \sigma_1 \) is annihilated in the next time. The rule in \( \sigma_r \) cannot be used because \( \sigma_r \) does not have any anti-spikes. At the same time, neuron \( \sigma_1 \) receives five spikes, among which two spikes are used to annihilate the two anti-spikes received from neuron \( \sigma_1 \); thus, the rule \( \lambda \rightarrow a \) in \( \sigma_1 \) can be applied. Neuron \( \sigma_1 \) receives one spike that annihilates one anti-spike \( \lambda \) received from neuron \( \sigma_1 \), and then the rule \( \lambda \rightarrow a \) in \( \sigma_1 \) is enabled to generate one spike \( a \).

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

(c) Module OUTPUT (Shown in Figure 5) Assume that \( \sigma_1 \) of system \( \prod \) accumulated one spike at time \( t \), and neuron \( \sigma_1 \) has \( n \) spikes for the number \( n \) being stored in register 1 of \( M \). When the rule in \( \sigma_1 \) is fired at time \( t \), neuron \( \sigma_1 \) sends one spike to \( \sigma_1 \). At this moment, \( \sigma_1 \) has an odd number of spikes and its rule fires. At time \( t + 1 \), \( \sigma_1 \) sends one spike to \( \sigma_{out} \) and \( \sigma_b \), respectively. Thus, neuron \( \sigma_{out} \) has one spike, which is an odd number. At time \( t + 2 \), neuron \( \sigma_{out} \) fires.
sending one spike to the environment. At the same time, the rules in \( \sigma_1 \) and \( \sigma_b \) are used, and both send one \( a \) to \( \sigma_{\text{out}} \). After \( n - 1 \) steps, until neuron \( \sigma_1 \) has no spike, the number of spikes in \( \sigma_{\text{out}} \) is even. At the same time, the use of the rule in \( \sigma_1 \) is stopped, and neuron \( \sigma_b \) has one spike. Neuron \( \sigma_{\text{out}} \) will receive one spike at time \( t + n + 2 \), and then the number of spikes is odd. Neuron \( \sigma_{\text{out}} \) fires a second time. Therefore, the number computed by the AWSN system is the difference between the first two steps when the neuron \( \sigma_{\text{out}} \) fires; that is, \( (t + n + 2) - (t + 2) = n \). The module \( \text{OUTPUT} \) can be simulated correctly.

4.2. Accepting Mode

**Theorem 2.** \( N_{ac} \text{ ASNPD}_s^2 = \text{NRE} \).

**Proof.** The proof of this theorem is similar to that of Theorem 1. A register machine \( M = (m, H, l_0, l_b, l) \), consisting of three modules, \( \text{ADD}, \text{SUB}, \) and \( \text{INPUT} \), is considered. Module \( \text{SUB} \) is shown in Figure 4.

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

(2) Module \( \text{INPUT} \) (Shown in Figure 7) Module \( \text{INPUT} \) shown in Figure 7 works as follows. The function of module \( \text{INPUT} \) is to read the spike train \( 10^{n-1} \) and compute the number \( n \) in the time between receiving two spikes. When neuron \( \sigma_{\text{in}} \) receives the first spike at time \( t \) and then neurons \( \sigma_d, \sigma_d' \), and \( \sigma_{\text{add}} \) receive one spike each, the rule in \( \sigma_d \) and \( \sigma_d' \) can be applied at time \( t + 1 \). At time \( t + 2 \), neuron \( \sigma_1 \) gets one spike, and, at the same time, neuron \( \sigma_{d_1} \) gets one spike from \( \sigma_{d_2} \) and neuron \( \sigma_{d_3} \) receives one \( a \) from \( \sigma_{d_2} \). Therefore, in the next \( n - 1 \) time periods, the rules in neurons \( \sigma_{d_2} \) and \( \sigma_{d_3} \) can be continued to be used.

During this period, \( \sigma_1 \) gets \( n - 1 \) spikes. When neuron \( \sigma_{\text{in}} \) receives the second spike at step \( t + n \), each of neurons \( \sigma_{d_1} \) and \( \sigma_{d_2} \) receives one spike at step \( t + n + 1 \) and then they both have two spikes. In this way, neurons \( \sigma_{d_2} \) and \( \sigma_{d_3} \) cannot fire to send any spikes to neuron \( \sigma_1 \). In the whole process, neuron \( \sigma_1 \) 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 \( \Pi^\prime \), used to simulate a universal register machine \( M^\prime \), is shown in Figure 8, which is a universal AWSN P system. \( \Pi^\prime \) 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 \( \text{INPUT} \) works as follows: when neuron \( \sigma_{\text{in}} \) gets a spike from the environment, the rule \( a \rightarrow a \) fires and one spike is sent to neurons \( \sigma_c, \sigma_c, \) and \( \sigma_c \), and two spikes are sent to neuron \( \sigma_c \). Then, the rule in neuron \( \sigma_c \) sends one spike to both \( \sigma_{c_1} \) and \( \sigma_1 \). At the same time, neuron \( \sigma_{c_2} \) fires and then sends one spike to \( \sigma_{c_1} \) and two spikes to \( \sigma_{c_3} \).
Up to this point, three spikes were sent to neuron $\sigma_c$. Therefore, before neuron $\sigma_{in}$ receives more spikes from the environment, neurons $\sigma_{c_1}$ and $\sigma_{c_2}$ have received one spike from each other in each time period and neuron $\sigma_l$ has received $g(x)$ spikes.

When $\sigma_{in}$ receives the second spike, each of the neurons $\sigma_{c_1}$, $\sigma_{c_2}$, and $\sigma_{c_3}$ can get one spike and $\sigma_{c_4}$ gets two spikes. Neuron $\sigma_{c_4}$ has four spikes at this moment, and its rule can be used to send two spikes to neuron $\sigma_{c_1}$. Neuron $\sigma_{c_1}$ then has six spikes, so that the rule in $\sigma_{c_1}$ is used to send one spike and send it to $\sigma_{c_2}$. In this way, neurons $\sigma_{c_2}$ and $\sigma_{c_4}$ receive one spike from each other in each step before $\sigma_{in}$ receives the third spike from the environment. Neuron $\sigma_{c_3}$ has $y$ spikes at the end. When neuron $\sigma_{in}$ receives the third spike, each of the neurons $\sigma_{c_1}$, $\sigma_{c_2}$, and $\sigma_{c_3}$ gets one spike, while $\sigma_{c_4}$ receives two spikes. As a result, neuron $\sigma_{c_4}$ has an odd number of spikes and the rule cannot be applied. At present, neuron $\sigma_{c_4}$ has three spikes, and the rule $a^3 \rightarrow a$ in neuron $\sigma_{c_4}$ fires, which generates one spike and sends it to $\sigma_{c_5}$. 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 $M'_w$. 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 $\sigma_{c_1}$ is used and $\sigma_{c_2}$ contains at least one spike, neuron $\sigma_{r_1}$ cannot fire. Neuron $\sigma_{r_1}$ fires by receiving one $\pi$ and then sends one spike to $\sigma_{c_2}$. At the end of the computation, neuron $\sigma_{c_1}$ has one spike, neuron $\sigma_{r_2}$ has one spike, and neuron $\sigma_{l_0}$ is empty. When $\sigma_{r_1}$ is empty, neurons $\sigma_{r_2}$ and $\sigma_{l_0}$ are also empty and neuron $\sigma_{l_0}$ contains one spike. Thus, each pair of SUB-ADD-1 instructions $l_i$: (SUB($r_{j_1}$),$l_{j_2}$) and $l_j$: (ADD($r_2$),$l_{j_2}$) can share a common neuron when $r_1 \neq r_2$, and there are totally 6 pairs in $M'_w$.
two instructions

12: Module ADD-ADD: the sequence of ADD and ADD instructions $l_{20}$: (ADD(0), $l_0$) and $l_{25}$: (SUB(3), $l_{18}$, $l_{20}$).

![Figure 11: Module SUB-ADD-2: the sequence of the ADD and SUB instructions $l_{20}$: (ADD(0), $l_0$) and $l_{25}$: (SUB(3), $l_{18}$, $l_{20}$).](image)

$\lambda$

![Figure 12: Module ADD-ADD: the sequence of ADD and ADD instructions $l_{17}$: (ADD(2), $l_{21}$) and $l_{21}$: (ADD(3), $l_{18}$).](image)

$\lambda$

![Figure 13: Module SUB-SUB with $r_1 \neq r_2$.](image)

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 $l_i$: (SUB($r_i$, $l_j$, $l_k$)) starts at time $t$. When neuron $\sigma_t$ gets a spike, the rule $a \rightarrow \bar{a}$ fires and sends one anti-spike to $\sigma_i$, and two anti-spikes to $\sigma_j$ and $\sigma_k$, respectively, at time $t + 1$. Neuron $\sigma_{c_i}$ receives an anti-spike at time $t + 2$. Neurons $\sigma_{r_i}$, $\sigma_{j_i}$, $\sigma_{k_i}$, and $\sigma_{c_i}$ work in the same way as those in module SUB shown in Figure 4. Neuron $\sigma_{c_i}$ will send three spikes to $\sigma_{c_i}$ and two spikes to $\sigma_{r_i}$, where forgetting rules will be applied. Thus, the instruction $l_i$: (SUB($r_i$, $l_j$, $l_k$)) 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 $l_i$: (SUB($r_i$, $l_j$, $l_k$)) is simulated and one spike is contained in neuron $\sigma_i$. The process is divided into two cases according to the number of spikes in neuron $\sigma_i$. When $\sigma_i$ has at least one spike, the working process of the system is similar to that of module SUB. When $\sigma_i$ is empty, the rule in neuron $\sigma_{c_i}$ cannot be used. Neurons $\sigma_{j_i}$, $\sigma_{k_i}$, and $\sigma_{c_i}$ are all empty but neuron $\sigma_{r_i}$ 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...
The string is read through neuron $\sigma_{in}$, 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 $\sigma_2$ 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 ($l_b$ 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 than SN 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, $a \rightarrow \pi$, $a \rightarrow a$, and $\pi \rightarrow a$, used that are time dependent, and there is one type of forgetting rules, $a^c \rightarrow \lambda$. 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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>$\sigma_{in}$</th>
<th>$\sigma_{f_1}$</th>
<th>$\sigma_{f_2}$</th>
<th>$\sigma_{f_3}$</th>
<th>$\sigma_{f_4}$</th>
<th>$\sigma_1$</th>
<th>$\sigma_2$</th>
<th>$\sigma_{out}$</th>
<th>$\sigma_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t + 1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t + 2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t + 3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$t + 4$</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t + 5$</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t + 6$</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 15: Module INPUT-OUTPUT.

$10^{g(x)-1}$ spikes are received by neuron $\sigma_1$. This system is then loaded with an arbitrary number $k$, and neuron $\sigma_2$ receives $k$ spikes. The number $k$ is also the output at the same time as the output spike train $10^{g(x)-1}$, 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 $M_b$ is simulated. If the computation in $M_b$ 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 $l_b$ 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 $\sigma_{in}$ has one spike at time $t$, and neuron $\sigma_{f_1}$ has two spikes. At time $t + 1$, $\sigma_{f_1}$ and $\sigma_{f_2}$ receive one spike, respectively. From the structure shown in Figure 15, neurons $\sigma_{f_3}$ and $\sigma_{f_4}$ receive one spike from each other at each step until $\sigma_{f_1}$ and $\sigma_{f_2}$ stop firing. Then $\sigma_{in}$ receives the second spike. Each of neurons $\sigma_1$ and $\sigma_2$ receives one spike, $\sigma_{f_3}$ receives six spikes, and $\sigma_{out}$ receives two spikes, so that neurons $\sigma_{f_3}$ and $\sigma_{out}$ can fire. At time $t + 3$, both $\sigma_{f_3}$ and $\sigma_{f_4}$ have two spikes, but they cannot fire again. $\sigma_{f_3}$ receives six spikes from $\sigma_{f_2}$, but $\sigma_{f_4}$ also receives two anti-spikes from $\sigma_{f_3}$, plus four spikes existing in $\sigma_{f_3}$, so that neuron $\sigma_{out}$ has eight spikes. In addition, neuron $\sigma_{out}$ receives two spikes again, so that there are three spikes contained. Neuron $\sigma_{f_3}$ only has one spike because the received anti-spike annihilates one spike. At time $t + 4$, the neuron $\sigma_{f_3}$ is empty after receiving an anti-spike. $\sigma_{f_3}$ receives two anti-spikes, so that there are four spikes contained in neuron $\sigma_{f_3}$, the number of spikes is even, and its rule can fire. At the next step, $\sigma_{f_4}$ receives one anti-spike and fires. Neuron $\sigma_{f_4}$ consumes two spikes and still can fire. At time $t + 6$, neurons $\sigma_{f_4}$ and $\sigma_{out}$ receive one spike from $\sigma_{f_5}$, respectively. So, there are 4 spikes in $\sigma_{out}$, meeting the required conditions for firing. Neuron $\sigma_{f_5}$ also gets one spike.
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 11CGJJ22), 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**


