Hindawi
Computational Intelligence and Neuroscience
Volume 2021, Article ID 3285719, 15 pages
https://doi.org/10.1155/2021/3285719

Research Article

Universal Nonlinear Spiking Neural P Systems with Delays and Weights on Synapses

Liping Wang, Xiyu Liu, and Yuzhen Zhao

Business School, Shandong Normal University, Jinan, China

Correspondence should be addressed to Xiyu Liu; xyliu@sdnu.edu.cn and Yuzhen Zhao; zhaoyuzhen@sdnu.edu.cn

Received 28 May 2021; Accepted 6 August 2021; Published 26 August 2021

Academic Editor: José Alfredo Hernández-Pérez

The nonlinear spiking neural P systems (NSNP systems) are new types of computation models, in which the state of neurons is represented by real numbers, and nonlinear spiking rules handle the neuron's firing. In this work, in order to improve computing performance, the weights and delays are introduced to the NSNP system, and universal nonlinear spiking neural P systems with delays and weights on synapses (NSNP-DW) are proposed. Weights are treated as multiplicative constants by which the number of spikes is increased when transiting across synapses, and delays take into account the speed at which the synapses between neurons transmit information. As a distributed parallel computing model, the Turing universality of the NSNP-DW system as number generating and accepting devices is proven. 47 and 43 neurons are sufficient for constructing two small universal NSNP-DW systems. The NSNP-DW system solving the Subset Sum problem is also presented in this work.

1. Introduction

Membrane computing (MC) is a representative of a new type of computing, abstracted from the phenomenon of signal transmission between cells in animals. It was proposed by Gheorghe Păun in 1998 and published in 2000 [1]. As a new type of natural computing, membrane computing has abundant model support [2–4] and is widely used in real life [5, 6]. The distributed computing model is named after the membrane system or P system. So far, P systems are mainly divided into three categories: cell-like P systems [7, 8], tissue-like P systems [9, 10], and neural-like P systems. Spiking neural P (SNP) systems [11], axon P systems [12], and dendrite P systems [13] are widely studied types of neural-like P systems. The research on SNP systems has been abundant for more than ten years. Similar to the spiking neural network (SNN), in the SNP system, neurons are activated, and the spikes are transmitted to other neurons along the synapse. SNP systems encode information through spikes in neurons. Intuitively, SNP systems are represented by a directed graph, where the nodes in the graph represent neurons, and the neurons are connected by arcs representing synapses. Neurons contain spikes and rules; two kinds of rules are used in SNP systems: firing rules $(E/a^r) \longrightarrow a$ and forgetting rules $a^s \longrightarrow \lambda$. Generating, accepting, and functional computing are the three working modes of SNP systems. It is worth mentioning that, by introducing neuron division, budding, or dissolution, the SNP system has been proven to be able to solve some computationally difficult problems, like SAT problem and Subset Sum problem [14, 15]. SNP can also solve many practical problems such as fault diagnosis of power systems [16–18], image processing [19, 20], and combination optimization problem [21]. Based on those innovative works, more and more scholars pay attention to SNP systems. Many variants of SNP systems have been proposed, and their computing power has also been proven.

The generation of the SNP system itself is affected by the spike signal in biological phenomena. From the perspective of biological facts, Păun et al. considered another important component related to brain function, astrocytes, which implicitly control the number of spikes along the axon. The functional application of astrocytes in the SNP model was first introduced in 2007 [22]. In 2012, Pan et al. formally proposed SNP systems with astrocytes [23]. More discussions about this type of system have been formed, such as

[24, 25]. For such systems, astrocytes influence computation by controlling the transmission of spikes on synapses. Considering the difference in the number of synapses connected between neurons, Pan et al. [26] previously proposed spiking neural P systems with weighted synapses. Then considering the mutual annihilation between spikes, Pan and Păun [27] used a pair of antispikes $(a, \overline{a})$ to explain the SNP system with antispikes. In the initial SNP system, there was a unique synapse connection between two neurons. Based on biological facts, Peng et al. [28] proposed that the synapse of each neuron can have an indefinite number of channels connected to subsequent neurons, and the SNP system with multiple channels was reasonably verified. Later, for the phenomenon that neurons carry positive and negative ions, the polarized SNP system was studied [29, 30]. In view of this neurophysiological observation, the speed of information transmission on the synapse is different. Song et al. [31] proposed spiking neural P systems with delay on synapses (SNP-DS) in 2020. On the other hand, driven by the ideas of mathematics and computing science, many variants of SNP systems make the SNP computing model with a parallel mechanism more powerful, for example, a complete arithmetic calculator constructed from SNP systems [32], SNP systems with random application of rules by introducing probability [33], homogeneous SNP systems with structural plasticity by adding and deleting connections between neurons [34], SNP systems with colored spikes by the idea of colored Petri nets [35], SNP systems with communication on request by using the parallel-cooperating grammar system to communicate on request [36], SNP systems with learning functions used to identify letters and symbols [37], and numerical SNP systems inspired by the numerical nature of the numerical P (NP) systems [38].

For SNP systems and variants mentioned above, the number of spikes in neurons is in integer form. So the nonlinear spiking neural P (NSNP) systems were investigated by Peng et al. [39]. In this system, the nonnegative integer state of neurons is replaced by real numbers; the number of spikes consumed and generated is replaced by nonlinear functions. However, NSNP systems need 117 and 164 neurons to construct Turing universal systems as functional computing device and number generator, respectively. For a new heuristic computing model such as this one, computing power is an important measure. At the same time, as a computational model of exchanging space for time, the computing power of NSNP systems needs to be further improved.

In this work, for the purpose of greater calculation power of NSNP systems, inspired by [26, 31], a novel P system, the nonlinear spiking neural P systems with delays and weights on synapses are proposed, abbreviated as NSNP-DW. In NSNP-DW systems, related to several factors, such as the dendrites' length, the spikes emitted from the same firing neuron reach different postsynaptic neurons at different times. So delays on synapses are introduced into the NSNP systems. Also, the number of dendrites among a firing neuron and its postsynaptic neurons is different. Although the spikes emitted from one neuron are the same, postsynaptic neurons with different amounts of dendrites to the firing neuron receive different amounts of spikes. Therefore, weights on synapses are added to the NSNP systems, and weights are treated as multiplicative constants by which the number of spikes varies when transiting across synapses. Both of these elements play a role in enhancing the computing power of the system.

Different from NSNP systems, the main novelties of this work are as follows:

(i) We propose a novel P system called universal nonlinear spiking neural P systems with delays and weights on synapses closer to biological neurons.

(ii) For the NSNP systems in [39], integer spikes are replaced by nonlinear functions. In the proposed NSNP-DW system, we find more applicable nonlinear functions that are also common in neural networks and machine learning, so as to abstract the complex responses generated by spikes.

(iii) We prove the computational power of the new P system; in addition, 47 and 43 neurons are successfully used for simulating function calculation and number generator, respectively.

(iv) Computational efficiency is also an important evaluation for a P system, usually considering whether the NP-complete problem can be solved in a feasible time. We prove the computational efficiency of the new P system by solving a typical NP-complete problem: Subset Sum problem in polynomial time. It is an attraction of P system solving computationally hard problems so quickly. This makes the NSNP-DW system another powerful tool to solve the NP-complete problem.

The remaining contributions of the paper are reflected as follows: Section 2 briefly shares the relevant content of register machine. Section 3 proposes NSNP-DW systems and gives two intuitive examples. The computing power of NSNP-DW systems equivalent to the Turing machine is proven in Section 4. Section 5 verifies the universality of the NSNP-DW system using fewer neurons. The uniform solution of the Subset Sum using the NSNP-DW system is added in Section 6. Conclusion and follow-up research on the NSNP-DW system are explained in the last section.

2. Prerequisites

The elements of the formal language of the SNP system can be consulted in [11, 26]. Here, we only introduce some symbolic theories used in this work, such as Turing machine construction, execution instructions, and universal Turing machine.

We show that the proposed NSNP-DW system as number generating and accepting device is equivalent to Turing machine. An arbitrary family of Turing computable natural numbers, defined as NRE, is a family of length sets for recursively enumerable languages. NRE can be characterized by a register machine M. The structure of the form $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H denotes the instruction tag set, l_0 is the starting label, l_h is the halting instruction HALT, and I denotes the instruction set. Each instruction in I is one of the following three forms:

(1) l_i: $(\mathrm{ADD}(r), l_j, l_k)$ (add 1 to register r and then execute nondeterministically the instruction with label l_j or l_k).

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

(3) l_h: HALT (the halting instruction).

By imitating the register machine, the universal NSNP-DW system was verified. When all the registers are empty, the calculation starts from l_0, and the instructions in I are continuously executed until halting instruction l_h. The number set generated by the register machine M is defined as N_{gen}. N_{acc} is the number set accepted by M; corresponding instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$ is deterministic and can be expressed as l_i: $(\mathrm{ADD}(r), l_j)$.

Computable function $f: N^k \longrightarrow N$ (k is a natural number) can be calculated by the register machine. If the equation $\psi_x(y) = M_u(g(x), y)$ is satisfied, where x and y are nonnegative integers and $g(x)$ is a recursive function, then the register machine M_u is universal. A universal register machine M_u' simulated by the NSNP-DW system is shown in Figure 1, consisting of 9 registers, 14 SUB instructions, 10 ADD instructions, and one HALT instruction.

3. NSNP-DW Systems

The materials and methods section should contain sufficient detail so that all procedures can be repeated. It may be divided into headed subsections if several methods are described.

In the following, we provide the definition of the NSNP-DW system and related semantic explanations. An example shows the operation of the system more clearly.

3.1. Definition. The structure of the proposed NSNP-DW system with degree $m \geq 1$ is

$$\Pi = \left(O, \sigma_1, \sigma_2, \ldots, \sigma_m, \mathrm{syn}, D_{\mathrm{syn}}, \mathrm{in}, \mathrm{out}\right), \tag{1}$$

where

(1) $O = \{a\}$ is the singleton alphabet (a denotes spike);

(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons, in the form of $\sigma_i = (x_i, R_i)$ for $1 \leq i \leq m$, where $x_i \in R^+$ is the initial value of spikes contained in neuron σ_i, indicating the initial state of neuron σ_i, and R_i is the finite set of spiking rules in the following form:

  (a) Spiking/firing rules: $E|a^{p(x_i)} \longrightarrow a^{q(x_i)}$, where E is the firing condition, $p(x_i)$ is a linear or nonlinear function, and $q(x_i)$ is a nonlinear function, and $p(x_i) \geq q(x_i) \geq 0$.

  (b) Forgetting rules: $E|a^{p(x_i)} \longrightarrow \lambda$, for a linear or nonlinear function $p(x_i)$.

(3) $\mathrm{syn} \in \{1, \ldots, h\} \times \{1, \ldots, h\} \times W$ is a synaptic expression with weight, where $W = \{1, \ldots, n\}$, $h = m \cup$ environment. For any $(i, j, n) \in \mathrm{syn}$, $1 \leq i, j \leq h$, $i \neq j$, and $n \in W$.

(4) D_{syn} represents the delay on synapse (i, j), expressed in the form of a time number.

(5) in and out indicate the input and output neurons of the system, respectively.

$l_0: (\mathrm{SUB}(1), l_1, l_2)$	$l_1: (\mathrm{ADD}(7), l_0)$
$l_2: (\mathrm{ADD}(6), l_3)$	$l_3: (\mathrm{SUB}(5), l_2, l_4)$
$l_4: (\mathrm{SUB}(6), l_5, l_3)$	$l_5: (\mathrm{ADD}(5), l_6)$
$l_6: (\mathrm{SUB}(7), l_7, l_8)$	$l_7: (\mathrm{ADD}(1), l_4)$
$l_8: (\mathrm{SUB}(6), l_9, l_0)$	$l_9: (\mathrm{ADD}(6), l_{10})$
$l_{10}: (\mathrm{SUB}(4), l_0, l_{11})$	$l_{11}: (\mathrm{SUB}(5), l_{12}, l_{13})$
$l_{12}: (\mathrm{SUB}(5), l_{14}, l_{15})$	$l_{13}: (\mathrm{SUB}(2), l_{18}, l_{19})$
$l_{14}: (\mathrm{SUB}(5), l_{16}, l_{17})$	$l_{15}: (\mathrm{SUB}(3), l_{18}, l_{20})$
$l_{16}: (\mathrm{ADD}(4), l_{11})$	$l_{17}: (\mathrm{ADD}(2), l_{21})$
$l_{18}: (\mathrm{SUB}(4), l_0, l_{22})$	$l_{19}: (\mathrm{SUB}(0), l_0, l_{18})$
$l_{20}: (\mathrm{ADD}(0), l_0)$	$l_{21}: (\mathrm{ADD}(3), l_{18})$
$l_{22}: (\mathrm{SUB}(0), l_{23}, l_{24})$	$l_{23}: (\mathrm{ADD}(8), l_{22})$
l_{24}: HALT	

FIGURE 1: The universal register machine M_u'.

The NSNP-DW system can be visualized by a directed graph G_Π with nodes and arcs, where nodes are neurons and arcs represent synapse connections. In original SNP systems, the number of spikes is described by an integer. In NSNP-DW systems, besides integer, it can also be described by nonlinear functions. For the rules $E|a^{p(x_i)} \longrightarrow a^{q(x_i)}$ and $E|a^{p(x_i)} \longrightarrow \lambda$ in the NSNP-DW system, the firing condition E has two forms. (1) It is a regular expression like $a(a^3)^+$ to exactly "cover" the contents of the neuron. If E is exactly $a^{p(x_i)}$, then E can be omitted. (2) It is a threshold for enabling the rule and exists in the form of a positive real number. In order to distinguish, we write the rules as $T|a^{p(x_i)} \longrightarrow a^{q(x_i)}$ and $T|a^{p(x_i)} \longrightarrow \lambda, T \in R^+$. Both types of rules can be used in this paper. We assume that $x_i(t)$ is the spike value of neuron σ_i at step t. For firing rules $T|a^{p(x_i)} \longrightarrow a^{q(x_i)}$, only when $x_i(t) \geq p(x_i)$ and $x_i(t) \geq T$ are satisfied, the rule can be applicable. Intuitively speaking, when the firing rule is met, the neuron is fired, consuming $p(x_i)$ spikes (if the remaining $(x_i(t) - p(x_i))$ spike can no longer enable the rule, it will disappear in the neuron) and sending out $q(x_i)$ spikes. Forgetting rules $T|a^{p(x_i)} \longrightarrow \lambda$, $q(x_i) \equiv 0$ mean that spikes with value $p(x_i)$ are consumed but not generated in the neuron to which it belongs. There will inevitably be multiple rules (greater than or equal to 2) in a neuron, assuming two rules, corresponding to the thresholds of T_1 and T_2, respectively. (i) When $T_1 \neq T_2$ and both rules can be executed, the maximum threshold strategy is applied. For instance, if $T_1 > T_2$, the rule with T_1 takes precedence, and the rule with threshold T_2 is not used. For this strategy, the forgetting rule is no exception. (ii) When $T_1 = T_2 = T$, nondeterministic rule selection strategy is enabled. That is, the rules with T_1 and T_2 need to be discussed separately.

In addition, for the NSNP-DW system, weights and delays are reflected in synapses. For any $(i, j, n) \in \mathrm{syn}$, n is the weight on the synapse. If the spikes with value $q(x_i)$ are emitted by neuron σ_i, neuron σ_j will receive $n \times q(x_i)$ spikes. The delay between synapses (i, j) is represented by D_{syn}, D_{syn} is in the form of a time number. If there is a delay between neurons σ_i and σ_j, the spiking rules $T|a^{p(x_i)} \longrightarrow a^{q(x_i)}$ belonging to neuron σ_i are enabled at step t. The spikes with value $p(x_i)$ are consumed from σ_i in step $t+1$; neuron σ_j will receive the spikes with value $q(x_i)$ from neuron σ_i at step $t + D_{\mathrm{syn}} + 1$. Therefore, the spikes are owned by σ_j after $D_{\mathrm{syn}}(i, j)$ moments.

Besides, the consumption function $p(x_i)$ and the production function $q(x_i)$ can be selected from the following common activation functions in neural networks:

(1) Tanh function: $f(x) = \tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$.

(2) Sigmoid function: $f(x) = 1/(1 + e^{-cx})$.

(3) RuLU function: $f(x) = \begin{cases} x, & x \geq 0, \\ 0, & x < 0. \end{cases}$

(4) Derivative function of RuLU: $f(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases}$

(5) ELU function: $f(x) = \begin{cases} x, & x \geq 0, \\ \alpha(e^x - 1), & x < 0. \end{cases}$

(6) $f(x) = \begin{cases} 2, & x > 0, \\ 0, & x \leq 0. \end{cases}$

(7) $f(x) = \begin{cases} 1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0. \end{cases}$

(8) $f(x) = \begin{cases} 1, & x > 1, \\ x, & -1 \leq x \leq 1, \\ -1, & x < -1. \end{cases}$

(9) LReLU function: $f(x) = \begin{cases} x, & x > 0, \\ \alpha x, & x \leq 0. \end{cases}$

(10) PReLU function: $f(x) = \max(\alpha x, x)$.

(11) Softplus function: $f(x) = \log(1 + e^x)$.

(12) Swish function: $f(x) = x \cdot \mathrm{sigmoid}(\beta x)$.

In the NSNP-DW system, neurons are used in parallel, and the use strategy of rules in each neuron is in a sequential manner; that is, only one rule is allowed to be employed nondeterministically in a calculation step.

Assuming m neurons, $x_i(t)$ is the number of spikes of the i-th $(1 \leq i \leq m)$ neuron, then the configuration (state) of the system Π at step t can be expressed as $(x_1(t), \ldots, x_m(t))$, and the initial configuration is $X_0 = (x_1(0), \ldots, x_m(0))$. By executing spike rules, configuration X_1 to configuration X_2, denoted by $X_1 \Rightarrow X_2$, is defined as a transition of system Π, and the sequence obtained by this transition is defined as a calculation. Each calculation is related to a spike sequence similar to a binary sequence. The sequence is composed of 0 and 1. The output neuron emits a spike to the environment corresponding to 1; otherwise, it corresponds to 0. In this study, the time interval at which the output neuron emits spikes to the environment is used as the calculation result.

For an NSNP-DW system with at most m neurons and at most 2 rules in each neuron, we use $\mathrm{N}_{\mathrm{gen}}\mathrm{SNP}_m^2$ to represent all natural number sets generated by the NSNP-DW system and $\mathrm{N}_{\mathrm{acc}}\mathrm{SNP}_m^2$ to represent all natural number sets accepted by the NSNP-DW system. When the number of neurons is uncertain, m is often replaced by *.

3.2. Two Illustrative Examples

Example 1. A simple example of the NSNP-DW system is given in Figure 2, containing three neurons labeled by 1, 2, and 3. The weight between neuron σ_1 and neuron σ_2 is 2. The delay between neuron σ_1 and σ_3 is $D_{\mathrm{syn}}(1,3) = 1$, denoted by $t = 1$. It is assumed here that $p(x)$ and $q(x)$ take (5) and (4) of the above function.

At step t, neuron σ_1 receives two spikes from the environment, its state is $x = 2$, and rule $2|a^x \longrightarrow a^{p(x)/2}$ is applied. Neuron σ_1 consumes two spikes and sends one spike to neurons σ_2 and σ_3 each (because $(p(x)/2) = 1$). At step $t+1$, neuron σ_2 receives two spikes due to weight 2. At this moment, neuron σ_3 is not fired because of $D_{\mathrm{syn}}(1,3) = 1$. At step $t+2$, neuron σ_3 receives two spikes, one from neuron σ_1 and one from neuron σ_2 (because $q(x) = 1$). There are two rules in neuron σ_3 that both meet the fired conditions. Subject to the maximum threshold strategy, rule $2|a^x \longrightarrow a^{q(x)}$ is used and emits one spike to the environment (for $q(x) = 1$). This example is complete and the results of each step are presented in Table 1.

Example 2. Let $p(x)$ and $q(x)$ take (2) and (3) of the above functions as the consumption and generation functions. We define Π_k as the system for generating natural numbers; as shown in Figure 3, each neuron initially has a spike.

In the first step, neuron σ_3 uses rule $1|a^{p(x)} \longrightarrow a^{q(x)/2}$ to emit 1/2 spike to the environment. At the same time, neurons σ_1 and σ_2 also fire by applying rule $1|a^{p(x)} \longrightarrow a^{q(x)/2}$, sending a spike to each other (because of weight 2), and neuron σ_1 and neuron σ_2 send one (because of weight 2) and 1/2 spike to neuron σ_3, respectively. In the next step, neurons σ_1 and σ_2 continue their initial actions any number of times, and the 3/2 spikes in neuron σ_3 are always removed. Once neuron σ_2 executes rule $1|a^{p(x)} \longrightarrow a^{\lambda}$ at step t, neuron σ_2 stops emitting spike, and neuron σ_1 sends the last spike to neuron σ_3. At step $t+1$, neuron σ_3 has a spike, using rule $1|a^{p(x)} \longrightarrow a^{q(x)/2}$ to send 1/2 spike to the environment again. In this way, the time interval $(t+1) - 1 = t$ of transmitting spike to the environment, that is, the natural number N generated by the system Π_k.

4. Computational Power

In the nonlinear spike rule of the NSNP-DW system, we choose two of the aforementioned functions (representing consumption or generation function) to verify the Turing universality of the system and its computing power. The following functions $p(x)$ and $q(x)$ are considered:

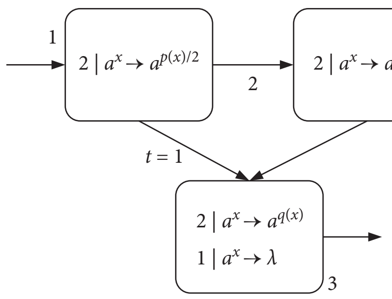


Figure 2: A simple example.

Table 1: Number of spikes at each moment in neurons.

Step	σ_1	σ_2	σ_3
t	2	0	0
$t+1$	0	2	0
$t+2$	0	0	2 (fire)

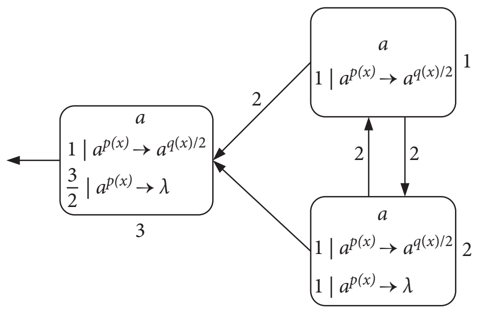

a
$1\,|\,a^{p(x)} \to a^{q(x)/2}$
1
2
a
$1\,|\,a^{p(x)} \to a^{q(x)/2}$
$\frac{3}{2}\,|\,a^{p(x)} \to \lambda$
3
2
2
a
$1\,|\,a^{p(x)} \to a^{q(x)/2}$
$1\,|\,a^{p(x)} \to \lambda$
2

Figure 3: An example of simulating natural number generation.

$$p(x) = \begin{cases} x, & x \geq 0, \\ \alpha\left(e^x - 1\right), & x < 0, \end{cases}$$
$$q(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{2}$$

Thus, the state of neuron σ_i can be recorded by a nonlinear equation:

$$x_i(t+1) = \begin{cases} x_i(t) - p\left(x_i(t)\right) + y_i(t), & \text{if neuron } \sigma_i \text{ fires}, \\ x_i(t) + y_i(t), & \text{otherwise}, \end{cases} \tag{3}$$

where $x_i(t)$ and $x_i(t+1)$ are the states of neuron at step t and $t+1$, respectively, $p(x_i(t))$ represents the consumption value, and $y_i(t)$ is the reception value.

In this part, we are committed to showing Turing universal NSNP-DW system as number generating device and accepting device. Given that the register machine M can generate and accept any NRE, the NSNP-DW system is proved to be universal through simulating the number generated by M. In order to facilitate understanding, we assume that number n in register r represents $2n$ spikes in neuron σ_r. Neurons σ_{l_i}, σ_{l_j}, and σ_{l_k} receive two spikes, and the corresponding instructions are activated.

4.1. NSNP-DW Systems as Number Generating Device

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

Proof. $\mathrm{N_{gen}SNP_*^2} \subseteq \mathrm{NRE}$ is beyond doubt based on the Turing-Church thesis; only $\mathrm{N_{gen}SNP_*^2} \supseteq \mathrm{NRE}$ needs to be proved. In the number generating mode, $M = (m, H, l_0, l_h, I)$ is the needed register machine, and the number generating device includes ADD, SUB, and FIN modules. M generates the number n in the following way: initially, the number of all registers is empty, and the simulation starts from instruction l_0, continues the process with the required label instructions, and stops at instruction l_h. According to the instructions, the number n stored in the first register is calculated by M. In ADD and SUB modules, neuron σ_{l_i} receives two spikes and runs according to instruction l_i: $(\mathrm{OP}(r), l_j, l_k)$ (OP is ADD or SUB operation). Neuron σ_{l_j} or σ_{l_k} receives two spikes indefinitely, and corresponding instruction l_j or l_k is activated. In the FIN module, neuron σ_{out} sends spikes outside twice at intervals.

ADD module (shown in Figure 4)—simulating an ADD instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$.

Neuron σ_{l_i} will receive two spikes from environment. After running the ADD mode, spikes will be sent to neuron σ_{l_j} or σ_{l_k} indefinitely to simulate instruction l_j or l_k. When two spikes are in neuron σ_r, the corresponding register r is increased by 1.

In detail, neuron σ_{l_i} fires at step t, and rule $2|a^x \longrightarrow a^{p(x)/2}$ is executed, sending one spike to neurons σ_{c_1}, σ_{c_2}, σ_{c_3}, and σ_r, respectively. The next moment, neuron σ_{c_1} receives one spike. Since the same thresholds of rule $1|a^x \longrightarrow a^{p(x)}$ and rule $1|a^x \longrightarrow \lambda$, the two rules are executed indefinitely:

(i) At step $t+1$, if rule $1|a^x \longrightarrow a^{p(x)}$ is used, one spike will be sent to neurons σ_{c_2} and σ_{c_3}, respectively. Next step, both neurons σ_{c_2} and σ_{c_3} contain two spikes, one from neuron σ_{l_i} and one from neuron σ_{c_1}. At step $t+3$, neuron σ_{c_3} fires by using rule $2|a^x \longrightarrow \lambda$, so that two spikes are removed. Rule $2|a^x \longrightarrow a^{p(x)/2}$ in neuron σ_{c_2} is applied simultaneously, consuming two spikes and emitting one to neurons σ_{c_4} and σ_{l_j}. At the next step, neuron σ_{c_4} becomes active by executing rule $1|a^x \longrightarrow a^{p(x)}$, emitting one spike to neuron σ_r and σ_{l_j} each. In this way, the second spike is received by σ_r, which will aggrandize the value of register r by 1. Neuron σ_{l_j} receives a total of two spikes successively, and then instruction l_j starts to be simulated.

(ii) At step $t+1$, neuron σ_{c_1} fires by using rule $1|a^x \longrightarrow \lambda$, which causes a spike to be removed. At step $t+2$, neurons σ_{c_2} and σ_{c_3} each receive one spike, and soon this no longer exists in neuron σ_{c_2} because of $1|a^x \longrightarrow \lambda$. The one received in neuron σ_{c_3} is sent to neurons σ_{c_4} and σ_{l_k} through rule $1|a^x \longrightarrow a^{p(x)}$. Then, the next moment, neuron σ_{c_4} transmits one, which is received by σ_r and σ_{l_k}. So neuron σ_r and σ_{l_k} have received two spikes at step $t+4$, respectively,

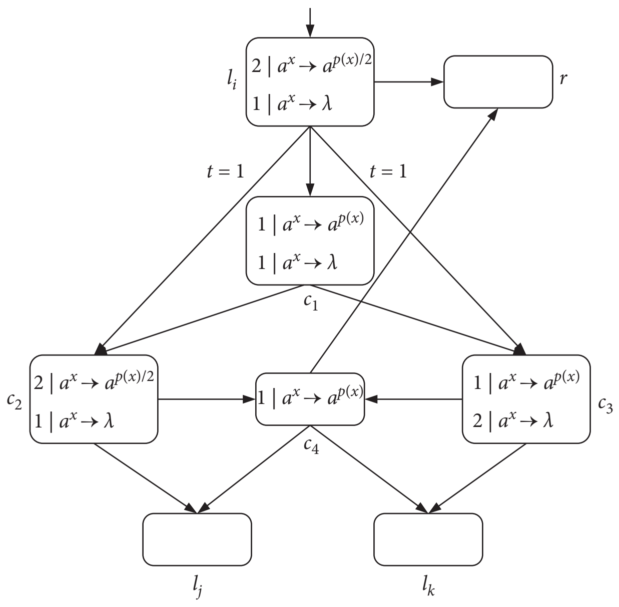

l_i
$2 \mid a^x \rightarrow a^{p(x)/2}$
$1 \mid a^x \rightarrow \lambda$
r
$t = 1$
$t = 1$
$1 \mid a^x \rightarrow a^{p(x)}$
$1 \mid a^x \rightarrow \lambda$
c_1
c_2
$2 \mid a^x \rightarrow a^{p(x)/2}$
$1 \mid a^x \rightarrow \lambda$
$1 \mid a^x \rightarrow a^{p(x)}$
c_4
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$
c_3
l_j
l_k

FIGURE 4: ADD module (simulating l_i: $(\mathrm{ADD}(r), l_j, l_k)$).

indicating that the register r is increased by 1, and l_k is activated.

Therefore, simulating instruction l_i: $(\mathrm{ADD}(r), l_j, l_k)$ is displayed correctly. Considering two different rules, Table 2 shows the number of spikes in all neurons at each moment.

SUB Module (shown in Figure 5)—simulating a SUB instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$.

Two spikes are received by neuron σ_{l_i}. If register r is not empty, then two spikes are sent to neuron σ_{l_j}, and the corresponding instruction l_j is executed. If the value in the register r is zero, then two spikes are sent to neuron σ_{l_k}, and the instruction with label l_k is executed. The detailed simulation process is as follows.

Neuron σ_{l_i} fires at step t, and rule $2|a^x \longrightarrow a^{p(x)/2}$ is applied to emit one spike. At step $t+1$, neuron σ_r, σ_{l_j}, and σ_{l_k} each receive one spike from neuron σ_{l_i}. Next, there will be two cases according to the value of spike in neuron σ_r:

(i) At step $t+1$, if $2n+1$ $(n \geq 1)$ spikes are contained by σ_r (the value of the corresponding register r is n), then rule $3|a^x \longrightarrow a^{q(x)}$ is applicable. Next step, the neuron σ_{c_1} receives three spikes and fires, one from neuron σ_{l_i} and two from neuron σ_r. Based on the maximum threshold strategy, rule $2|a^x \longrightarrow a^{\lambda}$ is used to consume these three spikes. At the same step, neuron σ_{l_j} receives the second spike, and then instruction l_j is simulated by system Π.

(ii) At step $t+1$, if neuron σ_r only contains one spike (the value of the corresponding register r is 0), then the spike is removed by rule $1|a^x \longrightarrow a^{\lambda}$. At step $t+2$, a spike from neuron σ_{l_i} is in neuron σ_{c_1}. The firing of neuron σ_{c_1} by rule $1|a^x \longrightarrow a^{p(x)}$ causes neuron σ_{l_k} to add a spike. At the next step, neuron σ_{l_k} receives a total of two spikes, and then instruction l_k is simulated by system Π, but not l_j.

So SUB module simulates instruction l_i: $(\mathrm{SUB}(r), l_j, l_k)$ correctly. The simulated numerical changes are presented in Table 3.

(3) FIN module (shown in Figure 6) - outputting the result of computation.

At step t, neuron σ_{l_h} fires by running rule $2|a^x \longrightarrow a^{p(x)/2}$, transmitting a spike to σ_1. Originally, neuron σ_1 contains $2(n-1)$ $(n \geq 2)$ spikes, and after receiving one spike, the rule $3|a^2 \longrightarrow a$ can be used first because of the maximum threshold strategy. Then neuron σ_{c_1} and neuron σ_{out} each have a spike from neuron σ_1. The first spike is sent by output neuron σ_{out} to the environment through $1|a^x \longrightarrow a^{p(x)}$ at step $t+3$. Besides, neuron σ_{out} receives one spike from neuron σ_1 and neuron σ_{c_1}, respectively, causing them to be forgotten by $2|a^x \longrightarrow \lambda$. Since two spikes are consumed in neuron σ_1, a spike is continuously given σ_{c_1} and σ_{out}. As a result, both spikes in neuron σ_{out} from $t+2$ to $t+n$ are forgotten by $2|a^x \longrightarrow \lambda$. Until step $t+n+1$, only one spike is kept in σ_1, and then the generated one is emitted by $1|a^x \longrightarrow a^{p(x)}$. At step $t+n+2$, the neuron σ_{out} still accepts two spikes but is forgotten instantly. At step $t+n+3$, the last spike is received by neuron σ_{out} from σ_{c_1} and sent to the environment through rule $1|a^x \longrightarrow a^{p(x)}$. The time interval between spikes emitted to the environment is the number calculated by the system Π. In short, the numerical result computed through the system Π is $(t+n+3) - (t+3) = n$. Take the generated number $n = 4$ as an example, and the simulated numerical changes of output module are reflected in Table 4.

Through the above discussion, we can see that the system Π accurately simulates the register machine M, so the theorem is reasonable. ☐

4.2. NSNP-DW Systems as Number Accepting Device

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

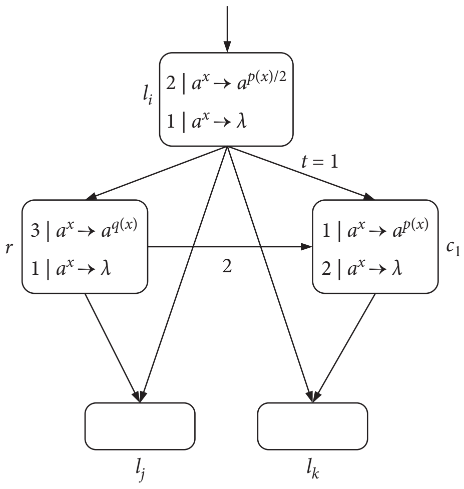
Proof. In the number accepting mode, the number in the first register is n (others are empty), and then the calculation starts from l_0; when the calculation stops, the number n is accepted. Similar to Theorem 1, we only need to verify $\mathrm{N_{acc}SNP^2_*} \supseteq \mathrm{NRE}$. The constructed NSNP-DW system as number accepting device includes an INPUT module, a deterministic ADD module, and a SUB module. Figure 7 shows the INPUT module.

Suppose that the first spike is received by neuron σ_{in} at step t; the firing of σ_{in} gives a spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}through rule $1|a^x \longrightarrow a^{p(x)}$. Then, neuron σ_{c_1} fires and outputs one spike to neuron σ_{l_0}, while neurons σ_{c_2} and σ_{c_3} fire by using rule $1|a^x \longrightarrow a^{p(x)}$. At step $t+2$, neurons σ_{c_2} and σ_{c_3} send one spike to each other, and especially neuron σ_1 receives two spikes from neuron σ_{c_2}, until neuron σ_{in} receives the second spike. Thus, neuron σ_1 receives $2n$ spikes from step $t+2$ to $t+n+1$.

At step $t+n+1$, neuron σ_{in} obtains a spike again and reacts using rule $1|a^x \longrightarrow a^{p(x)}$, sending one spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3} again. At step $t+n+2$, σ_{c_2} and σ_{c_3} each possess two spikes, and rule $2|a^x \longrightarrow \lambda$ is applicable so

TABLE 2: The spike result under ADD module.

Step	σ_{l_i}	σ_{c_1}	σ_{c_2}	σ_{c_3}	σ_{c_4}	σ_{l_j}	σ_{l_k}	σ_r								
t	2	0	0	0	0	0	0	$2n$								
$t+1$	0	1	0	0	0	0	0	$2n+1$								
	If $1\vert a^x \longrightarrow a^{p(x)}$ is used								If $1\vert a^x \longrightarrow \lambda$ is used							
	σ_{l_i}	σ_{c_1}	σ_{c_2}	σ_{c_3}	σ_{c_4}	σ_{l_j}	σ_{l_k}	σ_r	σ_{l_i}	σ_{c_1}	σ_{c_2}	σ_{c_3}	σ_{c_4}	σ_{l_j}	σ_{l_k}	σ_r
$t+2$	0	0	2	2	0	0	0	$2n+1$	0	0	2	2	0	0	0	$2n+1$
$t+3$	0	0	0	0	1	1	0	$2n+2$	0	0	0	0	1	0	1	$2n+1$
$t+4$	0	0	0	0	0	2	1	$2n+1$	0	0	0	0	0	1	2	$2n+2$
$t+5$	0	0	0	0	0	2	0	$2n+1$	0	0	0	0	0	0	2	$2n+2$

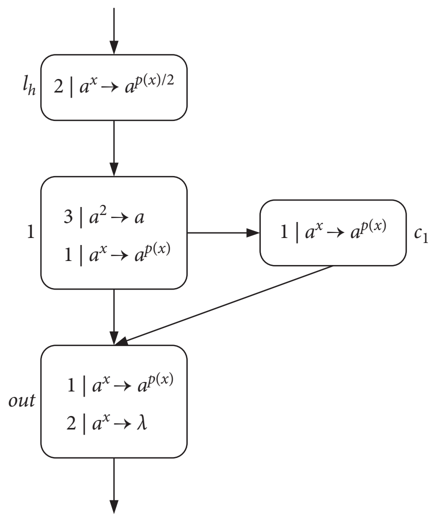


l_i
$2 \mid a^x \rightarrow a^{p(x)/2}$
$1 \mid a^x \rightarrow \lambda$
$t = 1$
r
$3 \mid a^x \rightarrow a^{q(x)}$
$1 \mid a^x \rightarrow \lambda$
2
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$
c_1
l_j
l_k

FIGURE 5: SUB module (simulating l_i: $(\mathrm{SUB}(r), l_j, l_k)$).

TABLE 3: The results of spike in SUB module.

	σ_r is not empty					σ_r is empty				
Step	σ_{l_i}	σ_{c_1}	σ_{l_j}	σ_{l_k}	σ_r	σ_{l_i}	σ_{c_1}	σ_{l_j}	σ_{l_k}	σ_r
t	2	0	0	0	$2n+1$	2	0	0	0	0
$t+1$	0	0	1	1	$2n+2$	0	0	1	1	1
$t+2$	0	3	2	1	$2n-1$	0	1	1	1	0
$t+3$	0	0	2	1	$2n-1$	0	0	1	2	0
$t+4$	0	0	2	0	$2n-1$	0	0	0	2	0

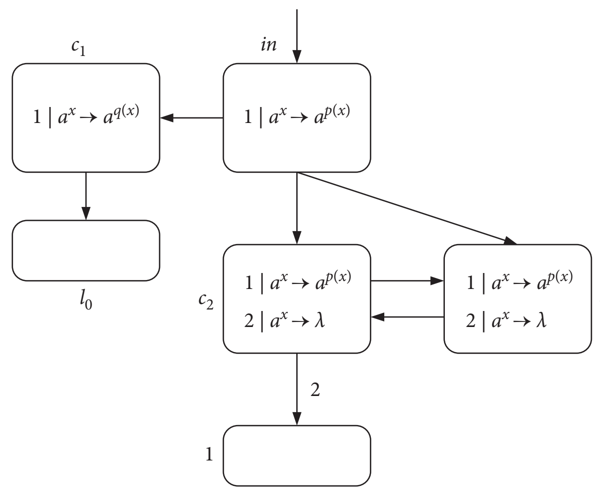


l_h
$2 \mid a^x \rightarrow a^{p(x)/2}$
1
$3 \mid a^2 \rightarrow a$
$1 \mid a^x \rightarrow a^{p(x)}$
$1 \mid a^x \rightarrow a^{p(x)}$
c_1
out
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$

FIGURE 6: FIN Module (output calculation result).

TABLE 4: The results of spike in FIN module.

Step	σ_{l_h}	σ_{c_1}	σ_1	σ_{out}	Environment
t	2	0	6	0	0
$t+1$	0	0	7	0	0
$t+2$	0	1	5	1	0
$t+3$	0	1	3	2	1
$t+4$	0	1	1	2	0
$t+5$	0	0	0	2	0
$t+6$	0	0	0	1	0
$t+7$	0	0	0	0	1



c_1
in
$1 \mid a^x \rightarrow a^{q(x)}$
$1 \mid a^x \rightarrow a^{p(x)}$
l_0
c_2
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$
c_3
2
1

FIGURE 7: INPUT module.

that spikes are eliminated. Simultaneously, neuron σ_{c_1} fires for the second step, executing rule $1|a^x \longrightarrow a^{q(x)}$ to give neuron σ_{l_0} a spike, whereupon instruction l_0 is activated.

Figure 8 displays the simulating of deterministic ADD instruction l_i: $(\mathrm{ADD}(r), l_j)$. Neuron σ_{l_i} accepts two spikes, consuming them and sending two spikes to neuron σ_{l_j} through rule $2|a^x \longrightarrow a^{p(x)}$; instruction l_j is simulated. Neuron σ_r contains two spikes, indicating that the register r is increased by 1. The SUB module of accepting mode is the same as in Figure 4.

In short, $\mathrm{N_{acc}SNP}^2_* = \mathrm{NRE}$ holds. □

5. Small Universal Computing Devices

5.1. Small Universal NSNP-DW Systems as Function Computing Device

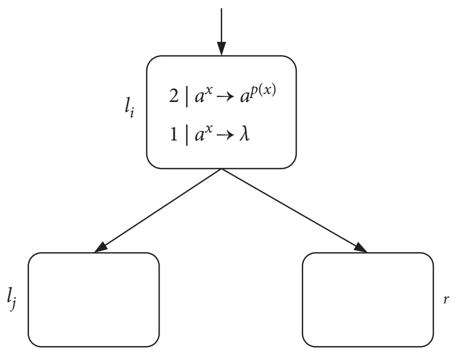


FIGURE 8: ADD module (simulating l_i: (ADD$(r), l_j$)).

Theorem 3. *There is a small universal NSNP-DW system possessing 47 neurons for computing functions.*

Proof. For simulation of the register machine M_u', the designed NSNP-DW system includes INPUT, ADD, SUB, and OUTPUT modules. The general design is visualized in Figure 9. Still the same as originally assumed, the value n in register r corresponds to $2n$ spikes in neuron σ_r. Assume that all neurons are empty in the initial configuration. Figure 10 is the designed INPUT module. σ_{in} is the input neuron, reading a spike train $10^{g(x)-1}0^{y-1}1$. Finally, $2g(x)$ and $2y$ spikes are contained by neurons σ_1 and σ_2, respectively.

As before, a time step or step represents the execution time of one rule. We still use this notion to mark the moment the rule is executed. At step t_1, if the first spike from the environment is received by neuron σ_{in}, rule $1|a^x \longrightarrow a^{p(x)}$ is applicable, sending one spike to neurons σ_{c_1}, σ_{c_2}, σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6}, respectively. At step t_1+1, neurons σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6} will not receive spikes due to one moment of delay; neuron σ_{c_1} and neuron σ_{c_2} each receive one spike but do not fire. At the next step, the neurons σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6} receive one spike from neuron σ_{in}, but neurons σ_{c_5} and σ_{c_6} do not fire. Neurons σ_{c_3} and σ_{c_4} fire by employing rule $1|a^x \longrightarrow a^{p(x)}$, sending one spike to each other and both sending one spike to σ_1, so neuron σ_1 owns two spikes at t_1+3. In this way, neuron σ_1 continues to receive two spikes, until step $t_1+g(x)+2$, σ_1 has a total of $2g(x)$ spikes.

At step t_2, here actually $t_2 = t_1 + g(x) + 2$, and neuron σ_{in} fires a second time and applies rule $1|a^x \longrightarrow a^{p(x)}$ to send one spike to each of neurons σ_{c_1}, σ_{c_2}, σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6}. Neurons σ_{c_1} and σ_{c_2} each have two spikes at step t_2+1. For one delay, neurons σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6}receive one spike from neuron σ_{in} at step t_2+2. At this step, neurons σ_{c_4} and σ_{c_5} each have two spikes, and executing rule $2|a^x \longrightarrow \lambda$ causes two spikes to be removed. On the contrary, neurons σ_{c_5} and σ_{c_6} each have two spikes and stay active. Two spikes are sent to each other by neurons σ_{c_5} and σ_{c_6}, and two are given to σ_2 by neuron σ_{c_5}, so that neuron σ_2 contains two spikes at step t_2+3. Neuron σ_2 accepts two spikes from neuron σ_{c_5} continuously until step t_2+y+3. At step t_2+y+3, neuron σ_2 retains $2y$ spikes in total.

At step t_3, here actually $t_3 = t_2 + y + 3$, neuron σ_{in} fires a third time, one spike is consumed, and one is sent to neurons σ_{c_1}, σ_{c_2}, σ_{c_3}, σ_{c_4}, σ_{c_5}, and σ_{c_6}, respectively, by rule $1|a^x \longrightarrow a^{p(x)}$. At step t_3+1, neuron σ_{c_1} with three spikes fires, and the rule $3|a^x \longrightarrow a^{2q(x)}$ is used to consume three spikes and send two spikes to neuron σ_{l_0}. When the neuron σ_{l_0} receives two spikes, the introduction l_0 is simulated. Neuron σ_{c_2} also fires at step t_3+1 and sends one spike to neurons σ_{c_3} and σ_{c_4} through rule $3|a^x \longrightarrow a^{q(x)}$. At step t_3+2, neuron σ_{c_3} receives spikes from neuron σ_{in} and neuron σ_{c_2}; forgetting rule $2|a^x \longrightarrow \lambda$ is applied to remove two spikes. The same is true for neuron σ_{c_4}. At the same step, neurons σ_{c_5} and σ_{c_6} each receive the third spike from neuron σ_{in}, so they have three spikes and remain inactive. The forgetting rule $3|a^x \longrightarrow \lambda$ is used to remove the three spikes.

In order to reflect the rationality of INPUT module, assuming $g(x) = 4$ and $y = 3$, the change in the number of spikes at each step can be clearly seen in Table 5.

In addition, the ADD and SUB modules are the same as in Figures 8 and 5. The design and simulation process of OUTPUT module is the same as Figure 6, except that the register 1 is replaced by register 8 (shown in Figure 11). This is because when a small universal NSNP-DW system is used as function computing device, after each instruction simulation, the final register 8 contains n numbers (the neuron σ_8 contains $2n$ spikes). The result n is output through the OUTPUT module.

From the discussion above, the NSNP-DW system as a function computing device can accurately simulate the register machine M_u' by using 57 neurons: (i) 7 neurons in the INPUT module, (ii) 2 neurons in the OUTPUT module, (iii) 1 neuron in each SUB instruction and 14 in total, (iv) 9 neurons in 9 registers, and (v) 25 neurons for 25 instructions.

The register machine M_u' is simulated by the NSNP-DW system, and each instruction l_i on M_u' is regarded as a neuron. However, some instructions are continuous. By exploring the relationship between the instructions of M_u', correspondingly constituted modules can be combined, instructions are omitted, and the use of neurons is reduced by the way. A detailed introduction to the initial register machine and its instructions can be found in [40]. For the NSNP-DW system, module combination is mainly divided into three categories: module ADD-ADD, module ADD-SUB, and module SUB-ADD (includes modules SUB-ADD-1 and SUB-ADD-2). The working process of module ADD-SUB and module SUB-ADD is closely related to that of module SUB. The working principle is expressed by the structure diagram. Readers interested in a description of the principle can refer to [11, 26, 41]. □

5.1.1. Module ADD-ADD

$$\begin{aligned} &l_{17}\colon\ (\text{ADD}(2), l_{21}),\\ &l_{21}\colon\ (\text{ADD}(3), l_{18}). \end{aligned} \tag{4}$$

These are two deterministic ADD instructions. The simulation of each instruction is the same as in Figure 8. The module design is shown in Figure 12 before the instruction l_{21} is omitted.

Obviously, this is a sequence of two consecutive ADD instructions connected by l_{21}; instruction l_{21} can be omitted through the following module ADD-ADD (shown in Figure 13).

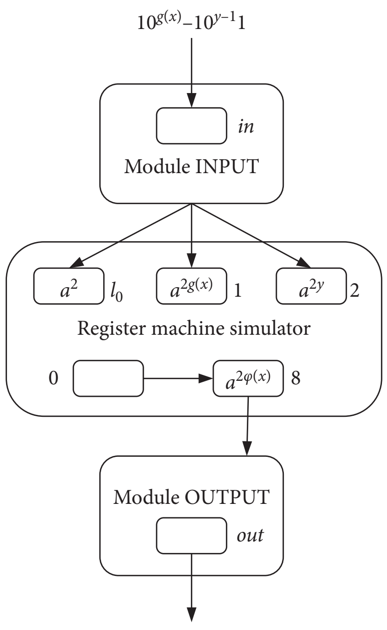


Figure 9: Framework of the universal NSNP-DW system.

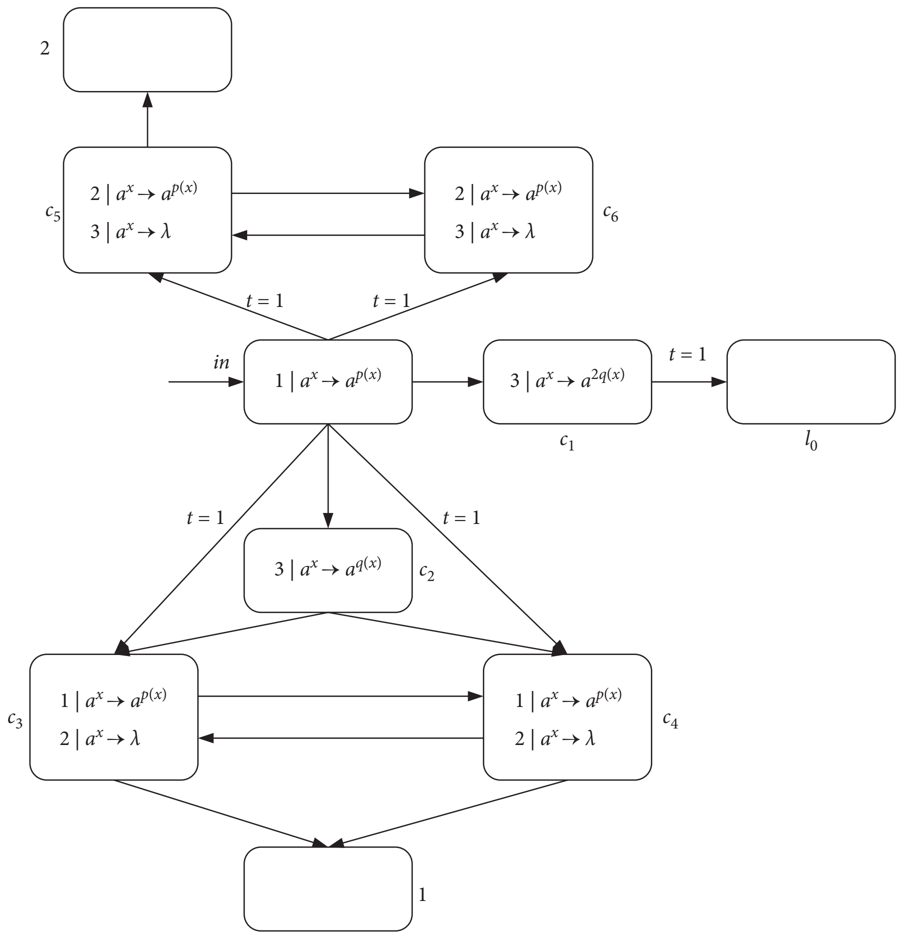

2
c_5
$2 \mid a^x \rightarrow a^{p(x)}$
$3 \mid a^x \rightarrow \lambda$
c_6
$t=1$
$t=1$
in
$1 \mid a^x \rightarrow a^{p(x)}$
$3 \mid a^x \rightarrow a^{2q(x)}$
c_1
$t=1$
l_0
$t=1$
$t=1$
$3 \mid a^x \rightarrow a^{q(x)}$
c_2
c_3
$1 \mid a^x \rightarrow a^{p(x)}$
$2 \mid a^x \rightarrow \lambda$
c_4
1

Figure 10: INPUT module.

TABLE 5: The results of spike in INPUT module when $g(x) = 4$ and $y = 3$.

Step	σ_{in}	σ_{c_1}	σ_{c_2}	σ_{c_3}	σ_{c_4}	σ_{c_5}	σ_{c_6}	σ_1	σ_2	σ_{l_0}
t	1	0	0	0	0	0	0	0	0	0
$t+1$	0	1	1	0	0	0	0	0	0	0
$t+2$	0	1	1	1	1	1	1	0	0	0
$t+3$	0	1	1	1	1	1	1	2	0	0
$t+4$	1	1	1	1	1	1	1	4	0	0
$t+5$	0	2	2	1	1	1	1	6	0	0
$t+6$	0	2	2	2	2	2	2	8	0	0
$t+7$	1	2	2	0	0	2	2	8	2	0
$t+8$	0	3	3	0	0	2	2	8	4	0
$t+9$	0	0	0	2	2	3	3	8	6	0
$t+10$	0	0	0	0	0	0	0	8	6	2

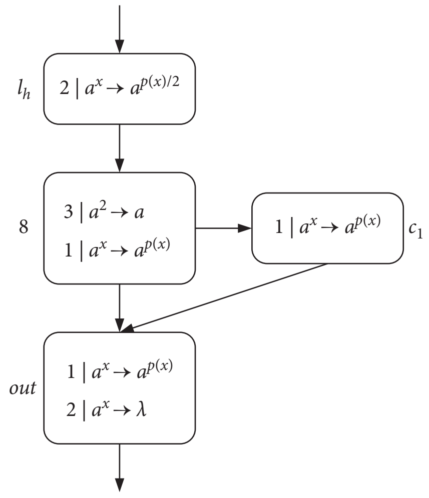

l_h
$2 \mid a^x \to a^{p(x)/2}$
8
$3 \mid a^2 \to a$
$1 \mid a^x \to a^{p(x)}$
$1 \mid a^x \to a^{p(x)}$
c_1
out
$1 \mid a^x \to a^{p(x)}$
$2 \mid a^x \to \lambda$

FIGURE 11: OUTPUT module.

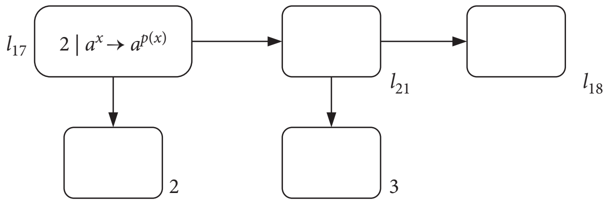


FIGURE 12: Module ADD-ADD before omitting l_{21}: the sequence of two consecutive ADD instructions l_{17}: (ADD(2), l_{21}) and l_{21}: (ADD(3), l_{18})

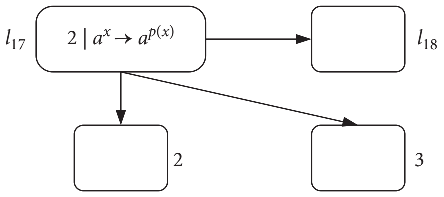


FIGURE 13: Module ADD-ADD after omitting l_{21}: the sequence of two consecutive ADD instructions l_{17}: (ADD(2), l_{21}) and l_{21}: (ADD(3), l_{18}).

5.1.2. Module ADD-SUB

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

These are two consecutive pairs of ADD-SUB instructions connected by l_6 and l_{10}, respectively; we can combine instructions l_5: (ADD(5), l_6) and l_6: (SUB(7), l_7, l_8) into one instruction l_5: (ADD(5), SUB(7), l_7, l_8), which saves instruction l_6. Instructions l_9: (ADD(6), l_{10}) and l_{10}: (SUB(4), l_0, l_{11}) are combined into one instruction l_9: (ADD(6), SUB(4), l_0, l_{11}), saving instruction l_{10}.

Taking the omission of l_6 as an example, neuron σ_{l_5} sends spikes to σ_5 and σ_{l_6}, and then neuron σ_{l_6} performs the simulation of instruction l_6. Here, neuron σ_{l_6} can be omitted, and neuron σ_{l_5} replaces σ_{l_6} to directly simulate the SUB instruction. It can be seen from Figure 14 that this is possible.

5.1.3. Module SUB-ADD. For introduction l_i: (SUB(r), l_j, l_k), when the value $r \neq 0$, it is subtracted by 1, and the instruction l_j is executed; otherwise, the labeled l_k is activated. Therefore, considering that there are two forms of consecutive SUB-ADD instructions, module SUB-ADD is divided into modules SUB-ADD-1 and SUB-ADD-2.

Module SUB-ADD-1:

$$\begin{aligned} &l_{15}\colon (\text{SUB}(3), l_{18}, l_{20}),\\ &l_{20}\colon (\text{ADD}(0), l_0). \end{aligned} \tag{6}$$

This is the case when l_k is activated and l_j is reserved. We can combine instructions l_{15}: (SUB(3), l_{18}, l_{20}) and l_{20}: (ADD(0), l_0) into one instruction l_{15}: (SUB(3), l_{18}, ADD(0), l_0), causing instruction l_{20} to be omitted (see Figure 15).

Module SUB-ADD-2:

$$\begin{aligned} &l_0\colon (\text{SUB}(1), l_1, l_2),\\ &l_1\colon (\text{ADD}(7), l_0),\\ &l_4\colon (\text{SUB}(6), l_5, l_3),\\ &l_5\colon (\text{ADD}(5), l_6),\\ &l_6\colon (\text{SUB}(7), l_7, l_8),\\ &l_7\colon (\text{ADD}(1), l_4),\\ &l_8\colon (\text{SUB}(6), l_9, l_0),\\ &l_9\colon (\text{ADD}(6), l_{10}),\\ &l_{14}\colon (\text{SUB}(5), l_{16}, l_{17}),\\ &l_{16}\colon (\text{ADD}(4), l_{11}),\\ &l_{22}\colon (\text{SUB}(0), l_{23}, l_{24}),\\ &l_{23}\colon (\text{ADD}(8), l_{22}). \end{aligned} \tag{7}$$

This is the case when l_j is activated and l_k is reserved. There are six pairs of instructions that can be combined in pairs. It is found through observation that the following

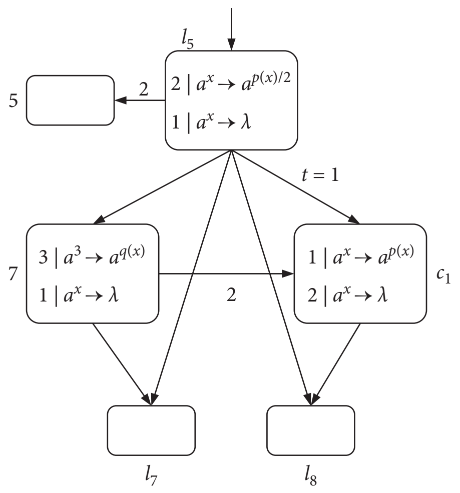


Figure 14: A module for consecutive ADD-SUB instruction l_5: (ADD(5), SUB(7), l_7, l_8).

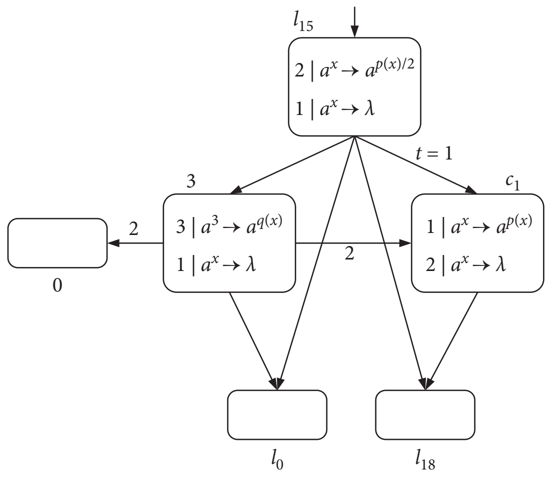


Figure 15: Module SUB-ADD-1: the sequence of two consecutive instructions l_{15}: (SUB(3), l_{18}, l_{20}) and l_{20}: (ADD(0), l_0).

ADD instruction happens to be the first execution position of the previous SUB instruction. Then each pair of SUB-ADD instruction combinations can be updated to l_i: (SUB(r_1), l_j, l_k) andl_j: (ADD(r_2), l_g). When the register $r_1 \neq r_2$, they can share one neuron σ_{l_j}. In this way, six neurons in total of σ_1, σ_5, σ_7, σ_9, σ_{19}, and σ_{23} can be saved. The visualization can be illustrated in Figure 16.

Through the above instruction combination (called "code optimization" in [41]), 10 neurons σ_{21}, σ_6, σ_{10}, σ_{20}, σ_1, σ_5, σ_7, σ_9, σ_{16}, and σ_{23} can be omitted. In the end, 47 neurons are enough to complete a small universal NSNP-DW system for computing functions:

(i) Nine neurons for 9 registers.
(ii) 15 neurons in remaining 15 instruction labels (ten labels are saved).
(iii) Seven neurons in the module INPUT.
(iv) 14 neurons for a total of 14 SUB instructions.
(v) Two neurons in the module OUTPUT.

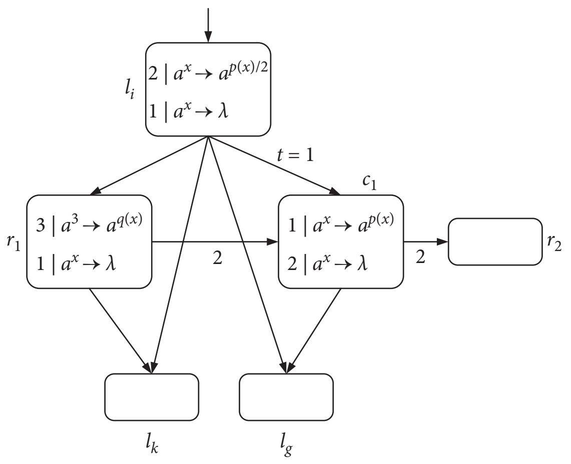


Figure 16: Module SUB-ADD-2: the sequence of the SUB and ADD instructions l_i: (SUB(r_1), l_j, l_k) and l_j: (ADD(r_2), l_g).

5.2. Small Universal NSNP-DW Systems as Number Generator. In the simulation of number generator, the INPUT module can be combined with the OUTPUT module, found in Figure 17. In the constructed INPUT-OUTPUT module, instruction l_h is removed, and register 8 is no longer needed. The spike train that the input neuron gets from the environment is $10^{g(x)-1}1$, neuron σ_1 is loaded with $2g(x)$ spikes, and neuron σ_2 receives $2n$ spikes nondeterministically. Neuron σ_{out} fires twice successively, and the time interval n is the numerical result generated.

The simulation of NSNP-DW systems as number generator shares 43 neurons, and the specific details will not be repeated:

(i) Eight neurons for 8 registers (no register 8).
(ii) 14 neurons in the remaining 14 instruction labels (l_h and ten labels are saved).
(iii) Seven neurons in the module INPUT-OUTPUT.
(iv) 14 neurons for a total of 14 SUB instructions.

Theorem 4. *There is a small universal NSNP-DW system possessing 43 neurons for number generator.*

The specific simulation will not be introduced in detail. We use an example to analyze the feasibility of number generator simulation; assuming $g(x) = 2$ and $n = 2$, the results of each step are reflected in Table 6.

5.3. Discussion. Theorem 3 gives the Turing universal NSNP-DW system with fewer neurons as a function computing device. In order to more intuitively verify the computing power of the NSNP-DW system, Table 7 compares the number of computing units for the variant and its related systems. According to Table 7, we observe that NSNP systems, SNP systems, SNP-DS systems, and recurrent neural networks use 117, 67, 56, and 886 neurons, respectively, to accomplish Turing universality for computing function, and NSNP-DW systems only require 47 neurons. Besides, according to Table 8, we have observed that 121

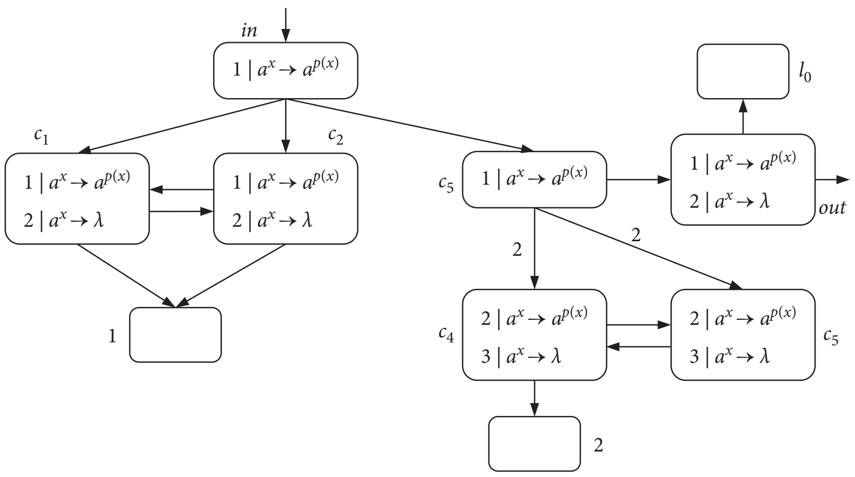

in
$1 \mid a^x \rightarrow a^{p(x)}$
c_1
c_2
$2 \mid a^x \rightarrow \lambda$
c_5
l_0
out
2
c_4
$3 \mid a^x \rightarrow \lambda$
$2 \mid a^x \rightarrow a^{p(x)}$
1
2

Figure 17: INPUT-OUTPUT module.

Table 6: The results of spike in INPUT-OUTPUT module.

Step	σ_{in}	σ_{c_1}	σ_{c_2}	σ_{c_3}	σ_{c_4}	σ_{c_5}	σ_1	σ_2	σ_{out}	σ_{l_0}
t	1	0	0	0	0	0	0	0	0	0
$t+1$	0	1	1	1	0	0	0	0	0	0
$t+2$	1	1	1	0	2	2	2	0	1 (fire)	0
$t+3$	1	2	2	1	2	2	4	2	0	1
$t+4$	0	0	0	0	4	4	4	4	1 (fire)	1
$t+5$	0	0	0	0	0	0	4	4	0	2

Table 7: Comparison of the least neurons of several calculation models.

Computing models	Number of neurons
NSNP-DW systems	47
NSNP systems [39]	117
SNP systems [11]	67
SNP-DS systems [31]	56
Recurrent neural networks [42]	886

Table 8: Comparison of the least neurons as number generator.

Computing models	Number of neurons
NSNP-DW systems	43
NSNP systems [39]	164

neurons are reduced for simulating number generator. In short, NSNP-DW systems are better than these systems in the use of neurons, and the computational power of the NSNP system has been effectively improved.

6. A Uniform Solution to Subset Sum Problem

The Subset Sum problem is one of the typical NP-complete problems proposed in [43]. We use the NSNP-DW system to solve the uniform solution of the Subset Sum in a nondeterministic operation mode.

The Swish function and the LReLU function are considered for the spike consumption function and the generating function in the problem.

$$\phi(x) = x \cdot \text{sigmoid}(x),$$
$$\gamma(x) = \begin{cases} x, & x > 0, \\ \alpha x, & x \le 0. \end{cases} \tag{8}$$

Problem. NAME: Subset Sum.

INSTANCE: a set of positive integers $V = \{v_1, v_2, \ldots, v_n\}$ and a positive integer S.

QUESTION: is there a subset $B \subseteq V$ that satisfies $\sum_{b \in B} b = S$?

Theorem 5. *The uniform solution of Subset Sum problem can be solved by NSNP-DW systems.*

Figure 18 depicts the architecture of the NSNP-DW system to solve the Subset Sum in a uniform way. The complexity of the uniform solution is that the system only "recognizes" the number n when solving the problem. The instance of the problem needs to be introduced into the system. $\sigma_{g_{i,3}}$ $(1 \le i \le n)$ is the input neuron of the system. The positive integer v_i $(1 \le i \le n)$ in the problem is encoded into $\sigma_{g_{i,3}}$. At the beginning of the calculation, $3(v_1 - 1)$ spikes $(a^{3(v_1-1)})$ enter neuron $\sigma_{g_{1,3}}$, $3(v_2 - 1)$ spikes $(a^{3(v_2-1)})$ enter neuron $\sigma_{g_{2,3}}$, ..., and $3(v_n - 1)$ spikes $(a^{3(v_n-1)})$ enter neuron $\sigma_{g_{n,3}}$. In the initial configuration (state) of the system, except that neuron σ_i $(1 \le i \le n)$ contains four spikes, all other neurons are empty.

In the first calculation, both rules in neuron σ_i are likely to be employed first (because they have the same threshold). The indeterminate use of these two rules indicates that the system solves this Subset Sum problem in a nondeterministic way of operation, and it also corresponds to whether the integer v_i is in the subset B. In the following, we carry out a complete derivation.

Proof. Neuron σ_i initially has four spikes. At step one, if rule $4|a^{\phi(x)} \longrightarrow a^{(3/4)\gamma(x)}$ is selected for use, then neuron σ_i will consume $4 \cdot \text{sigmoid}(4)$ spikes and send three spikes to neurons $\sigma_{g_{i,1}}$ and $\sigma_{g_{i,2}}$, respectively (because $\gamma(x) = 4$). At step 2, neuron $\sigma_{g_{i,1}}$ forgets three spikes by the rule $3|a^{\phi(x)} \longrightarrow \lambda$. At the same time, neuron $\sigma_{g_{i,2}}$ uses rule

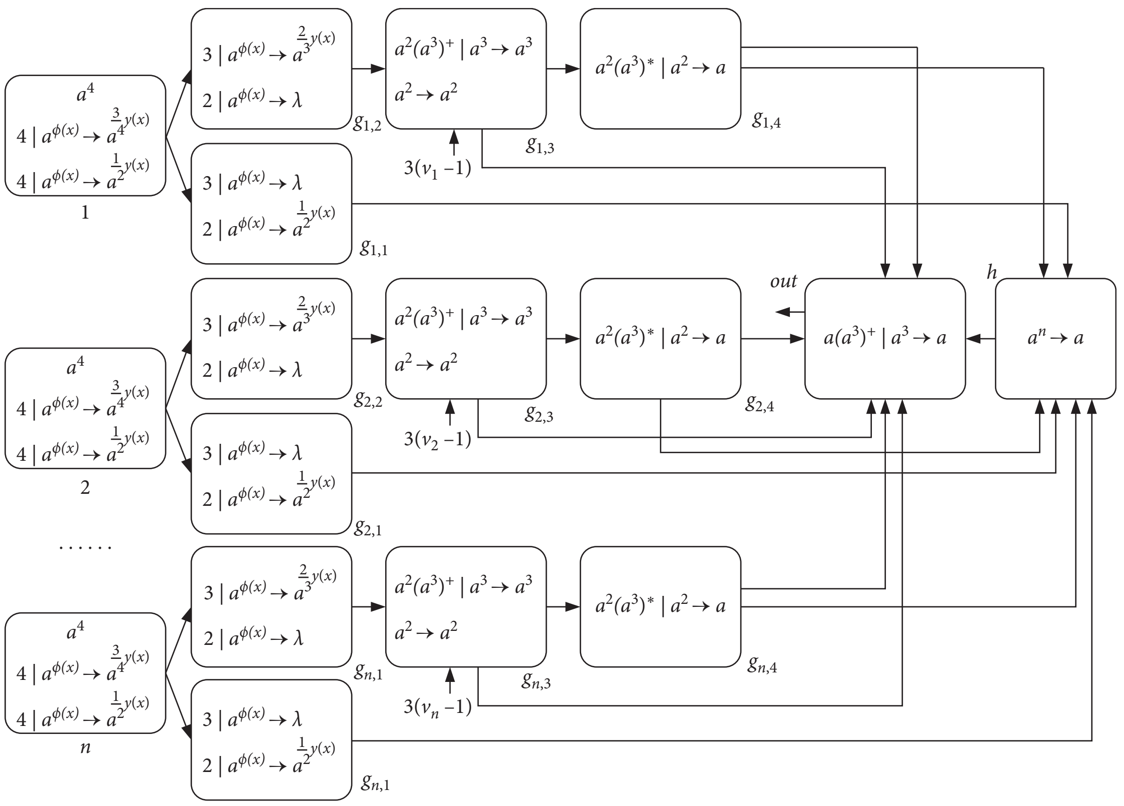

a^4
$4 \mid a^{\phi(x)} \to a^{\frac{3}{4}\gamma(x)}$
$4 \mid a^{\phi(x)} \to a^{\frac{1}{2}\gamma(x)}$
1
2
n
$3 \mid a^{\phi(x)} \to a^{\frac{2}{3}\gamma(x)}$
$2 \mid a^{\phi(x)} \to \lambda$
$3 \mid a^{\phi(x)} \to \lambda$
$2 \mid a^{\phi(x)} \to a^{\frac{1}{2}\gamma(x)}$
$a^2(a^3)^+ \mid a^3 \to a^3$
$a^2 \to a^2$
$a^2(a^3)^* \mid a^2 \to a$
$a(a^3)^+ \mid a^3 \to a$
$a^n \to a$
$g_{1,1}$ $g_{1,2}$ $g_{1,3}$ $g_{1,4}$
$g_{2,1}$ $g_{2,2}$ $g_{2,3}$ $g_{2,4}$
$g_{n,1}$ $g_{n,3}$ $g_{n,4}$
$3(v_1-1)$
$3(v_2-1)$
$3(v_n-1)$
out
h
......

FIGURE 18: An NSNP-DW system solving the Subset Sum problem.

$3|a^{\phi(x)} \longrightarrow a^{(2/3)\gamma(x)}$ to become active and sends two spikes to neurons $\sigma_{g_{i,3}}$ (because $\gamma(x) = 3$). At the end of this step, neurons $\sigma_{g_{i,1}}$ and $\sigma_{g_{i,2}}$ maintain their original state. At step 3, neuron $\sigma_{g_{i,3}}$ has a total of $3v_i - 1$ spikes and fires. The rule $a^2(a^3)^+|a^3 \longrightarrow a^3$ is used first, sending three spikes to neurons $\sigma_{g_{i,4}}$ and σ_{out}, respectively. This process will continue for $v_i - 1$ steps until the rule $a^2(a^3)^+|a^3 \longrightarrow a^3$cannot be activated. Neurons $\sigma_{g_{i,4}}$ and σ_{out} still cannot be active after receiving $3k$ ($k \in N$) spikes. When there are only two spikes left in neuron $\sigma_{g_{i,3}}$, rule $a^2 \longrightarrow a^2$ is used, and finally, two spikes are sent to neuron $\sigma_{g_{i,4}}$ and σ_{out}, respectively. After possessing $3k + 2$ spikes, neuron $\sigma_{g_{i,4}}$ fires and emits a spike to neurons σ_{out} and σ_h, respectively. In the next step, the neuron σ_{out} still cannot fire because it takes $3k + 1$ spikes to be activated. Conversely, neuron σ_h that has received n spikes is activated by rule $a^n \longrightarrow a$ and sends one spike to neuron σ_{out}. In this way, the output neuron σ_{out} can fire and emit spikes to the environment.

If initially neuron σ_i uses the rule $4|a^{\phi(x)} \longrightarrow a^{(1/2)\gamma(x)}$, $4 \cdot \text{sigmoid}(4)$ spikes are consumed and two spikes are sent to neurons $\sigma_{g_{i,1}}$ and $\sigma_{g_{i,2}}$, respectively (because $\gamma(x) = 4$). In the second step, the two spikes received by neuron $\sigma_{g_{i,2}}$ are removed by rule $2|a^{\phi(x)} \longrightarrow \lambda$. Neuron $\sigma_{g_{i,1}}$ uses the rule $2|a^{\phi(x)} \longrightarrow a^{(1/2)\gamma(x)}$ and sends a spike to neuron σ_h (because $\gamma(x) = 2$). Before the neuron σ_h fires, neuron σ_{out} remains inactive. After neuron σ_h receives a total of n spikes from $\sigma_{g_{i,1}}$ $(1 \le i \le n)$, it fires and sends a spike to neuron σ_{out}. In the next step, the neuron σ_{out} contains only one spike, so it does not fire, nor does it emit spikes into the environment.

At this point, we have ended the process of solving the uniform solution of the Subset Sum. Obviously, the system requires a total of $5n + 2$ neurons. After stopping operation, if there are exactly S spikes in the environment, the answer to the problem is "yes," which means that there is a subset $B \subseteq V$ that makes $\sum_{b \in B} b = S$ hold. Otherwise, it is "no." This is enough to show that the NSNP-DW system for solving Subset Sum problem is complete.

In the calculation process, the calculation between neurons is parallel, and the rules in each neuron are calculated sequentially. Through computing and reasoning, it can be known that neurons σ_i, $\sigma_{g_{i,1}}$, $\sigma_{g_{i,2}}$, $\sigma_{g_{i,4}}$, and σ_h fire once, respectively, and neuron $\sigma_{g_{i,3}}$ fires $\sum_{i=1}^{n} v_i$ times. After all other neurons stop computing, the neuron σ_{out} can fire at most $\sum_{i=1}^{n} v_i$ times. Therefore, the system needs $\sum_{i=1}^{n} 2v_i + 5$ steps to complete the calculation. In addition, we choose nonlinear functions as the spike consumption function and generation function, which is closer to reality and reflects the significance of nonlinear functions in the NSNP-DW system. □

7. Conclusions and Further Work

The nonlinear spiking neural P (NSNP) systems are variants of spiking neural P (SNP) systems. Nonlinear functions are used flexibly in NSNP systems. We focus on the computing power of NSNP systems in this work. Two mechanisms of delays and weights are introduced, and nonlinear spiking neural P systems with delays and weights (NSNP-DW) are proposed. An explicit example is given to visually demonstrate the operation of the NSNP-DW system. Through a series of simulation computing, 47 and 43 neurons are sufficient for constructing small universal NSNP-DW systems as function computing device and number generator. Compared with the NSNP systems [39], the NSNP-DW system decreases 70 neurons and 121 neurons, respectively, as function computing device and number generator. Finally, the uniform solution of the Subset Sum problem is solved efficiently by using the NSNP-DW system.

For further work, the NSNP-DW system, as a distributed parallel computing model, can be combined with clustering algorithms to improve algorithm efficiency. As far as the computational power of the NSNP-DW system is concerned, we are committed to proving that the 47 and 43 neurons used by the simulating function calculation and the number generator, respectively, are the least in total. In particular, the number of spikes breaks through the integer limit and has been replaced by nonlinear functions in NSNP systems. In view of this breakthrough, we can try to link NSNP-DW systems with the neural network to expand more interesting 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 project was supported by the National Natural Science Foundation of China (61876101, 61802234, and 61806114), Social Science Fund Project of Shandong Province, China (16BGLJ06 and 11CGLJ22), Natural Science Fund Project of Shandong Province, China (ZR2019QF007), Postdoctoral Project, China (2017M612339 and 2018M642695), Humanities and Social Sciences Youth Fund of the Ministry of Education, China (19YJCZH244), and Postdoctoral Special Funding Project, China (2019T120607).

References

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

[2] G. Zhang, Z. Shang, S. Verlan et al., "An overview of hardware implementation of membrane computing models," *ACM Computing Surveys*, vol. 53, no. 5, pp. 1–38, 2020.

[3] X. Liu and Q. Ren, "Spiking neural membrane computing models," *Processes*, vol. 9, no. 5, 2021.

[4] G. Zhang, M. J. Pérez-Jiménez, A. Riscos-Nuñez et al., *Membrane Computing Models: Implementations*, Springer, New York, NY, USA, 2021.

[5] 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.

[6] G. Zhang, M. J. Pérez-Jiménez, and M. Gheorghe, *Real-life Applications with Membrane Computing*, Springer, New York, NY, USA, 2017.

[7] B. Song, L. Pan, and M. J. Pérez-Jiménez, "Cell-like P systems with channel states and symport/antiport rules," *IEEE Transactions on NanoBioscience*, vol. 15, no. 6, pp. 555–566, 2016.

[8] 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.

[9] R. Freund, G. Păun, and M. J. Pérez-Jiménez, "Tissue P systems with channel states," *Theoretical Computer Science*, vol. 330, no. 1, pp. 101–116, 2005.

[10] A. Leporati, L. Manzoni, G. Mauri, A. E. Porreca, and C. Zandron, "Tissue P systems with small cell volume," *Fundamenta Informaticae*, vol. 154, no. 1-4, pp. 261–275, 2017.

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

[12] C. Haiming, T.-O. Ishdorj, and G. Paun, "Computing along the axon," *Progress in Natural Science*, vol. 17, no. 4, pp. 417–423, 2007.

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

[14] L. Pan, G. Păun, and M. J. Pérez-Jiménez, "Spiking neural P systems with neuron division and budding," *Science China-Information Sciences*, vol. 54, no. 8, pp. 1596–1607, 2020.

[15] Y. Zhao, X. Liu, and W Wang, "Spiking Neural P Systems with neuron division and dissolution," *PLoS One*, vol. 11, Article ID e0162882, 2016.

[16] T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, and M. J. Perez-Jimenez, "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.

[17] H. Peng, J. Wang, P. Shi, M. J. Pérez-Jiménez, and A. Riscos-Nunez, "Fault diagnosis of power systems using fuzzy tissue-like P systems," *Integrated Computer-Aided Engineering*, vol. 24, no. 1, pp. 401–411, 2017.

[18] H. Peng, J. Wang, J. Ming et al., "Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4777–4784, 2018.

[19] 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.

[20] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, and H. Peng, "Membrane computing and image processing: a short survey," *Journal of Membrane Computing*, vol. 1, no. 1, pp. 58–73, 2019.

[21] 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. 1440006–1440016, 2014.

[22] G. Păun, "Spiking neural P systems with astrocyte-like control," *Journal of Universal Computer Science*, vol. 13, no. 11, pp. 1707–1721, 2007.

[23] L. Pan, J. Wang, and H. J. Hoogeboom, "Spiking neural P Systems with astrocytes," *Neural Computation*, vol. 24, no. 3, pp. 805–825, 2012.

[24] 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.

[25] B. Aman and G. Ciobanu, "Spiking neural P systems with astrocytes producing calcium," *International Journal of Neural Systems*, vol. 30, 2020.

[26] 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.

[27] 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.

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

[29] T. Wu, A. Păun, 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, 2017.

[30] T. Wu, T. Zhang, and F. Xu, "Simplified and yet Turing universal spiking neural P systems with polarizations optimized by anti-spikes," *Neurocomputing*, vol. 414, pp. 255–266, 2020.

[31] X. Song, L. Valencia-Cabrera, H. Peng, J. Wang, and M. J. Perez-Jimenez, "Spiking neural P systems with delay on synapses," *International Journal of Neural Systems*, vol. 31, no. 1, 2020.

[32] G. Zhang, H. Rong, P. Paul, Y. He, F. Neri, and M. J. Pérez-Jiménez, "A complete arithmetic calculator constructed from spiking neural P systems and its application to information fusion," *International Journal of Neural Systems*, vol. 31, no. 1, pp. 1–17, 2021.

[33] P. P. L. Lazo, F. G. C. Cabarle, H. N. Adorna, and J. M. C. Yap, "A return to stochasticity and probability in spiking neural P systems," *Journal of Membrane Computing*, vol. 3, no. 2, pp. 149–161, 2021.

[34] R. T. A. de la Cruz, F. G. C. Cabarle, I. C. H. Macababayao, H. N. Adorna, and X. Zeng, "Homogeneous spiking neural P systems with structural plasticity," *Journal of Membrane Computing*, vol. 3, no. 1, pp. 10–21, 2021.

[35] T. Song, A. Rodriguez-Paton, P. Zheng, and X. Zeng, "Spiking neural P systems with colored spikes," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 4, pp. 1106–1115, 2018.

[36] L. Pan, G. Păun, G. Zhang, and F. Neri, "Spiking neural P systems with communication on request," *International Journal of Neural Systems*, vol. 27, no. 8, Article ID 1750042, 2017.

[37] 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.

[38] T. Wu, L. Pan, Q. Yu, and C. Tan, "Numerical spiking neural P systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 10, 2020.

[39] H. Peng, Z. Lv, B. Li et al., "Nonlinear spiking neural P systems," *International Journal of Neural Systems*, vol. 30, no. 10, Article ID 2050008, 2020.

[40] M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Hoboken, NJ, USA, 1967.

[41] A. Păun and G. Păun, "Small universal spiking neural P systems," *Bio Systems*, vol. 90, no. 1, pp. 48–60, 2007.

[42] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[43] R. G. Michael and S. J. David, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, WH Freeman & Company, New York, NY, USA, 1979.