Article

Spiking Neural P Systems with Rules Dynamic Generation and Removal

Yongshun Shen and Yuzhen Zhao *

College of Business, Shandong Normal University, Jinan 250014, China; 2022021067@stu.sdu.edu.cn
* Correspondence: zhaoyuzhen@sdu.edu.cn

Abstract: Spiking neural P systems (SNP systems), as computational models abstracted by the biological nervous system, have been a major research topic in biological computing. In conventional SNP systems, the rules in a neuron remain unchanged during the computation. In the biological nervous system, however, the biochemical reactions in a neuron are also influenced by factors such as the substances contained in it. Based on this motivation, this paper proposes SNP systems with rules dynamic generation and removal (RDGRSNP systems). In RDGRSNP systems, the application of rules leads to changes of the substances in neurons, which leads to changes of the rules in neurons. The Turing universality of RDGRSNP systems is demonstrated as a number-generating device and a number-accepting device, respectively. Finally, a small universal RDGRSNP system for function computation using 68 neurons is given. It is demonstrated that the variant we proposed requires fewer neurons by comparing it with five variants of SNP systems.

Keywords: spiking neural P systems; membrane computing; Turing universality

1. Introduction

Membrane computing (MC) [1], proposed in 1998, is a distributed parallel computational method inspired by the working process of biological cells. The computational models of MC are also known as membrane systems or P systems. P systems can be roughly classified into three groups according to the biological cells they imitate: cell-like P systems, tissue-like P systems, and neural-like P systems.

In neural-like P systems, spikes are used to represent information. Two classes of neural-like P systems have been proposed: axon P systems (AP systems) [2] and spiking neural P systems (SNP systems) [3]. An AP system has a linear structure, so that each node of this system can only send spikes to its neighboring left and right nodes. An SNP system has a directed graph structure, where neurons are represented as the nodes in the directed graph and the synapses between neurons are represented by directed arcs.

SNP systems, which also belong to the spiking neural network [4], have become a major research issue as soon as they were proposed. The research related to SNP systems mainly includes four aspects: variant model design, computational power proof, algorithm design (application), and implementation. Theoretical research mainly includes variant model design and computational power proof, while algorithmic research mainly includes algorithm design (application) and implementation. Under the continuous research of a wide range of scholars, several different variants of SNP systems have been proposed. Numerical SNP (NSNP) systems [5–7] are a variant of SNP systems inspired by numerical P systems, in which information is encoded by the values of variables and processed by continuous functions. Compared to the original SNP systems, NSNP systems are no longer discrete, but have a continuous numerical nature, which is useful for solving real-world problems. Homogeneous SNP (HSNP) systems [8–11] are a restricted variant of SNP systems in which each neuron has the same set of rules, making HSNP systems simpler than the original SNP systems. SNP systems with communication on request (SNQP...
systems) [12–16] are proposed by adding a communication strategy to the original SNP systems in which neuron request spikes from neighboring neurons according to the number of spikes it contains, and no spikes are consumed or produced during the computation. SNP systems with anti-spikes [17–20] are proposed by considering the anti-spikes and annihilation rules in the original SNP systems. In SNP systems with anti-spikes, the usual spikes and anti-spikes will annihilate each other when they meet in the same neuron. SNP systems with weights [21–25] are proposed by introducing synaptic weights to the original SNP systems in which the number of spikes received by a neuron depends on not only the number of spikes sent by its presynaptic neurons, but also the synaptic weights between neurons.

In addition, as a kind of bio-inspired computational model, SNP systems have many excellent characteristics. First, neurons are not fully connected to each other; therefore, the topological structure of SNP systems is sparse and simple. In SNP systems, information is transmitted only when meeting a specific value, so their information transmission is characterized by low power consumption. Additionally, because SNP systems are a kind of parallel computational model, the tasks can be processed in parallel, reducing the processing time required to be consumed significantly. In addition, the working mechanism of SNP systems is special. In SNP systems, spikes are used to represent information, and the information is processed and transmitted by using spiking rules. In SNP systems, a neuron can contain multiple rules and use regular expression as a control condition for the rules, allowing the neuron to freely select rules to use depending on the number of spikes. It is because of these characteristics that SNP systems have good performance in many kinds of applications [26,27], such as time series forecasting [28–31], pattern recognition [32,33], image processing [34–37], etc.

Generally, the rules of neurons in SNP systems are set in the initial state and cannot be changed during the computation process. However, in biological neurons, biochemical reactions are usually changed according to factors such as different substances in the neuron. Motivated by this, spiking neural P systems with rules dynamic generation and removal (RDGRSNP systems) is proposed. In RDGRSNP systems, rules in the rule set of a neuron can be added and removed during the computation process.

Our contribution is to propose a new variant of SNP systems called RDGRSNP systems. This new variant enriches the research on SNP systems. In RDGRSNP systems, the application of rules causes the number of spikes in the neuron to change, which causes the rules in the neuron to change. This makes the rules in RDGRSNP systems no longer constant like the traditional SNP systems, but dynamically changing during the computation. This is not considered by previous theoretical studies on SNP systems. Therefore, our proposed RDGRSNP systems are closer to biological nervous systems and fit better with biological facts than the original SNP systems. In addition, our proposed RDGRSNP systems can improve the ability of adaptive regulation of SNP systems based on dynamical change of rules according to the different substances in neurons.

This paper will be structured as follows: in Section 2, we introduce the proposed RDGRSNP systems and an illustrative example; in Section 3, we demonstrate the Turing universality of RDGRSNP systems when used as a number-generating device and a number-accepting device, respectively. In Section 4, a small universal RDGRSNP system for simulating function computation is constructed. In Section 5, the work of this paper is summarized.

2. SNP Systems with Rules Dynamic Generation and Removal

This section presents the definition of SNP systems with rules dynamic generation and removal and explains them with an illustrative example.

2.1. Definition and Description

In this subsection, we introduce the formal definition of RDGRSNP systems using abstract notation and a detailed description of the definition.
2.1.1. Definition

An RDGRSNP system $\Pi$ of degree $m$ consists of the following tuples:

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_n, R, \text{syn}, \text{in}, \text{out})$$

where

1. $O$ is a single alphabet with only one element $a$ used to represent a spike.
2. $\sigma_1, \sigma_2, \ldots, \sigma_n$ denote $m$ neurons in $\Pi$, and $\sigma_i$ is represented by tuple $\sigma_i = (n_i, R_i)$, where
   a. $n_i$ denotes the number of spikes in neuron $\sigma_i$.
   b. $R_i$ denotes the rule set contained in neuron $\sigma_i$, where $R_i \subseteq R$.
3. $R$ is the rule set of $\Pi$, which contains rules of the following two forms:
   a. $r_1: E/a^c \rightarrow a^p; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$. A rule of this form is called the spiking rule with rules dynamic generation and removal, where $E$ is a regular expression defined on the single alphabet $O$, and $c$ and $p$ denote the number of spikes consumed and produced with $c \geq p$, $r_1, r_2, \ldots, r_i$ denoting the $l$ rules in $R$, and $a_1^1, a_2^2, \ldots, a_i^i \in \{+,-,0\}$ are the labels. The spiking rule with rules dynamic generation and removal $r_1: E/a^c \rightarrow a^p; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$ can be applied when the number of spikes in the neuron satisfies conditions $a^c_i \in L(E)$ and $a^p_i \geq a^c_i$. Furthermore, the spiking rule with rules dynamic generation and removal can be abbreviated as $r_1: a^c \rightarrow a^p; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$ when $a^p \in L(E)$.
   b. $r_1: a^c \rightarrow \lambda; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$. A rule of this form is called the forgetting rule with rules dynamic generation and removal, where $s \geq 0$ denotes the number of spikes consumed, with the additional restriction that for all the spiking rules with rules dynamic generation and removal $r_1: E/a^c \rightarrow a^p; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$, there is $a^c \notin E$. Similarly, $r_1, r_2, \ldots, r_i$ denotes the $l$ rules $R$, and $a_1^1, a_2^2, \ldots, a_i^i \in \{+,-,0\}$ are the labels.
4. $\text{syn} = \{(i, j)\}$ denotes the set of synapses, where $(i, j) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ and $i \neq j$.
5. $\text{in, out} \in \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ denotes the input and output neurons, respectively.

2.1.2. Description

The definition of RDGRSNP systems proposed above is described here in detail. We mainly describe the process of applying rules in RDGRSNP systems, how the rules are executed in RDGRSNP systems, and the state and graphical definition of RDGRSNP systems.

When neuron $\sigma_i$ applies a spiking rule with rules dynamic generation and removal, $r_1: E/a^c \rightarrow a^p; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$, $c$ spikes in $\sigma_i$ are consumed, and $p$ spikes are sent along synapse $(i, j)$ to postsynaptic neuron $\sigma_j$. Meanwhile, neuron $\sigma_i$ adjusts its rule set $R_i$ as follows. When $a_k = +, a_i^r_k$ can be abbreviated to $r_k$, indicating that rule $r_k$ is added to the rule set $R_i$ of neuron $\sigma_i$. If rule $r_k$ already exists in $R_i$, no rule is repeatedly added. When $a_k = -$, it means that rule $r_k$ is removed from the rule set $R_i$ of neuron $\sigma_i$. If rule $r_k$ does not exist in $R_i$, no rule is removed. When $a_k = 0, a_i^r_k$ can be omitted and not written, and it means that rule $r_k$ is neither added to nor removed from the rule set $R_i$ of neuron $\sigma_i$.

When neuron $\sigma_i$ applies a forgetting rule with rules dynamic generation and removal $r_1: a^c \rightarrow \lambda; a_1^r_1, a_2^r_2, \ldots, a_i^r_i$, $s$ spikes in $\sigma_i$ are removed and no spikes are sent. Meanwhile, $\sigma_i$ adjusts its rule set, operating in the same way as when applying the spiking rule with rules dynamic generation and removal.

Note that an RDGRSNP system $\Pi$ is parallel on the system level and sequential on the neuron level. There is a global clock used to mark steps throughout the system. When a neuron contains only one applicable rule, the neuron must apply it. When a neuron contains two or more applicable rules, the neuron can only non-deterministically choose one to apply. When no rule is applicable in the neurons of $\Pi$, the system stops computing.
In general, the state of the system at each step is represented by the number of spikes in each neuron at that step. However, in the proposed RDGRSNP system \( \Pi \), since rules can be generated and removed dynamically, the rules contained in each neuron should also be included in the state of system \( \Pi \). That is, the state of \( \Pi \) is represented by the number of spikes and the rules in each neuron, symbolized as \( C_t = (k_1, R_1), (k_2, R_2), \ldots, (k_m, R_m) \). The initial state of \( \Pi \) is represented by \( C_0 \). When the number of spikes and the rule sets contained in the neurons of \( \Pi \) do not change, the system stops computing.

Regarding the graphical definition, rounded rectangles are used to represent neurons, and directed line segments are used to represent synapses between neurons. In an RDGRSNP system, the result of a computation is defined by the time interval between the first two spikes sent to the environment \( \text{env} \) by the output neuron \( \sigma_{\text{out}} \).

2.2. An Illustrative Example.

In Figure 1, an illustrative example is given to elaborate the definition of RDGRSNP systems. In this example, both \( \sigma_1 \) and \( \sigma_2 \) initially have one spike, and \( \sigma_{\text{out}} \) initially has two spikes, so the initial state is \( C_0 = ((1, \{ r_3 \}), (1, \{ r_3, r_4 \}), (2, \{ r_1 \})) \).

![Figure 1. An Illustrative Example.](image)

At step 1, neuron \( \sigma_1 \) applies \( r_3 : a \rightarrow a \) to send one spike to \( \sigma_2 \) and \( \sigma_{\text{out}} \). The output neuron \( \sigma_{\text{out}} \) applies the spiking rule with rules dynamic generation and removal \( r_1 : a^2 \rightarrow a; \neg r_1, r_2, r_3 \). Consumes two spikes, sends the first spike to \( \text{env} \), removes \( r_1 \), and adds \( r_2 \) and \( r_3 \). As for neuron \( \sigma_2 \), the number of spikes contained can satisfy the two rules \( r_2 \) and \( r_4 \), so it will non-deterministically choose one rule to apply.

Suppose at step 1, neuron \( \sigma_2 \) applies \( r_3 : a \rightarrow a \) and sends one spike to \( \sigma_1 \) and \( \sigma_{\text{out}} \). At step 2, \( \sigma_1 \) applies \( r_3 : a \rightarrow a \) again and sends one spike to \( \sigma_2 \) and \( \sigma_{\text{out}} \), respectively. As for the output neuron \( \sigma_{\text{out}} \), it contains two spikes and rules \( r_2 \) and \( r_3 \), so it applies rule \( r_2 : a^2 \rightarrow \lambda \), forgetting the two included spikes. Thereafter, if neuron \( \sigma_2 \) continues to apply rule \( r_3 : a \rightarrow a \), the system will continue to repeat the cycle until neuron \( \sigma_2 \) chooses rule \( r_4 : a \rightarrow \lambda \) to apply.

Suppose at step \( n \), where \( n \geq 1 \), neuron \( \sigma_2 \) chooses rule \( r_4 : a \rightarrow \lambda \) to apply, so it does not send spikes outward. In addition, neuron \( \sigma_1 \) sends one spike to \( \sigma_{\text{out}} \). Therefore, at step \( n + 1 \), \( \sigma_{\text{out}} \) has only one spike, applies rule \( r_3 : a \rightarrow a \), and sends the second spike to \( \text{env} \).

Thus, \( \sigma_{\text{out}} \) sends the first two spikes with a time interval of \( (n + 1) - 1 = n \), where \( n \geq 0 \), i.e., all positive integers.

3. The Turing Universality of SNP Systems with Rules Dynamic Generation and Removal

This section is used to prove the Turing universality of RDGRSNP systems when used as a number-generating device and a number-accepting device, respectively. It has been shown that register machines can be used to characterize Turing computable sets of numbers (NRE), so we demonstrate the Turing universality of RDGRSNP systems by simulating register machines.
For a register machine $M = (m, H, l_0, l_h, I)$, $m$ denotes the number of registers $r$ in the register machine $M$, $H$ is a label set in which each label corresponds to an instruction in the instruction set $I$, $l_0$ denotes the start instruction, $l_h$ denotes the halt instruction, and $I$ denotes an instruction set. The instructions in $I$ have the following three forms:

1. $l_i: (ADD(r), l_j, l_k)$, the instruction indicates that: add one to the number in $r$ and then jump non-deterministically to $l_j$ or $l_k$;
2. $l_i: (SUB(r), l_j, l_k)$, the instruction indicates that: if the number in $r$ is not zero, subtract the number by one and jump to $l_j$; if the number in $r$ is zero, jump directly to $l_k$;
3. $l_h: HALT$, the instruction is the halt instruction, indicating that $M$ stops computing.

In the number-generating mode, all registers in $M$ are initially empty, and $M$ starts from $l_0$ and executes until $l_h$. After the computation is finished, all registers except register 1 are empty, and the number generated by $M$ is stored in register 1. It is assumed that there is no $l_i: (SUB(r), l_j, l_k)$ for the instructions related to register 1, i.e., the number in register 1 will not be subtracted.

In the number-accepting mode, all registers of $M$ except register 1 are initially empty, and the number to be accepted is stored in register 1. If $M$ can reach the halt instruction $l_h: HALT$, it proves that the number is accepted by $M$. Note that the instruction $l_i: (ADD(r), l_j, l_k)$ needs to be changed to deterministic $l_i: (ADD(r), l_j)$ in the number accepting mode.

In this paper, we use $N_2RDGRSNP(\Pi)$ and $N_{RDGRSNP}(\Pi)$ to denote the set of numbers that can be generated and accepted by a RDGRSNP system, respectively.

In the proofs of this paper, we use neuron $\sigma_{h_0}$ to denote instruction $l_0$, neuron $\sigma_{r_j}$ to denote register $r$, and auxiliary neurons $\sigma_{h_{l_j}}$ for module construction, where $j \in N^+$. In addition, we stipulate that the simulation of instruction $l_i$ starts when $\sigma_{h_0}$ has four spikes.

**Theorem 1.** $N_2RDGRSNP(\Pi) = NRE$

**Proof.** By Turing Church’s theorem, there is $N_2RDGRSNP(\Pi) \subseteq NRE$, so it is sufficient if we can prove $NRE \subseteq N_2RDGRSNP(\Pi)$. For $NRE \subseteq N_2(\Pi)$, we simulate the register machine to prove it by following the process shown in Figure 2. $\square$

![Figure 2. The flowchart of the SNP system with rules dynamic generation and removal simulating register machine in the number-generating mode.](image)

In the initial state, all neurons are empty except neuron $\sigma_{h_0}$. Four spikes in neuron $\sigma_{h_0}$ are used to trigger the simulation of computation. The simulation of computation follows the simulation of the instructions in the register machine until neuron $\sigma_{h_i}$ receives four spikes. When neuron $\sigma_{h_i}$ receives four spikes, it means that the computation process...
in the register machine is successfully simulated, and the FIN module starts to output the computation result stored in register 1.

**ADD module:** The ADD module is used to simulate instruction $l_1 : (ADD(r), l_j, l_k)$, as shown in Figure 3. This module consists of six neurons and eight rules.

![Figure 3. ADD module.](image)

Suppose that at step $t$, $\sigma_l$ has four spikes and starts simulating instruction $l_1 : (ADD(r), l_j, l_k)$. In neuron $\sigma_l$, the spiking rule with rules dynamic generation and removal $r_1 : a^4 / a^2 \rightarrow a^2 ; r_2$ is applied, consuming two spikes, sending two spikes to $\sigma_r$, $\sigma_{l_1}^{(1)}$, and $\sigma_{l_2}^{(2)}$ and adding rule $r_2 : a^2 \rightarrow a^2 ; -r_2$. At step $t + 1$, $\sigma_l$ has two spikes with $r_1$, $r_2$, and $r_7$. Therefore, $\sigma_l$ applies rule $r_2 : a^2 \rightarrow a^2 ; -r_2$, sending two spikes to $\sigma_r$, $\sigma_{l_1}^{(1)}$, and $\sigma_{l_2}^{(2)}$ and removing rule $r_2$. At step $t + 1$, $\sigma_l$ receives two spikes again, for a total of four spikes, corresponding to the number $r$ add one. In addition, $\sigma_{l_1}^{(1)}$ has four spikes with rules $r_1$ and $r_2$. Since both rules can be applied, neuron $\sigma_{l_1}^{(1)}$ non-deterministically chooses one to apply. The situations are as follows:

1. When neuron $\sigma_{l_1}^{(1)}$ chooses rule $r_1 : a^4 / a^2 \rightarrow a^2$; $r_2$ to apply, it consumes two spikes, sends two spikes to neurons $\sigma_7$ and $\sigma_{l_2}^{(2)}$, and adds rule $r_2 : a^2 \rightarrow a^2 ; -r_2$. At step $t + 3$, $\sigma_{l_2}^{(2)}$ applies the forgetting rule with rules dynamic generation and removal $r_3 : a^6 \rightarrow \lambda ; r_6$, consumes the six spikes it contains, and adds rule $r_6 : a^2 \rightarrow \lambda ; -r_6$. In neuron $\sigma_{l_1}^{(1)}$, since it contains two spikes with rules $r_1$, $r_2$, and $r_3$, rule $r_2 : a^2 \rightarrow a^2 ; -r_2$ is applied, sending two spikes to neurons $\sigma_7$ and $\sigma_{l_2}^{(2)}$, and removing rule $r_2$. At step $t + 4$, $\sigma_{l_2}^{(2)}$ applies $r_6 : a^2 \rightarrow \lambda ; -r_6$, consuming two spikes and removing rule $r_6$. In addition, neuron $\sigma_l$ receives two spikes, containing four spikes in total and starts simulating $l_j$.

2. When neuron $\sigma_{l_2}^{(2)}$ chooses rule $r_3 : a^4 / a^2 \rightarrow \lambda ; r_4$ to apply, it consumes two spikes and adds rule $r_4$. At step $t + 3$, $\sigma_{l_1}^{(1)}$ contains two spikes with rules $r_1$, $r_3$, and $r_4$. Thus, the spiking rule with rules dynamic generation and removal $r_4 : a^2 \rightarrow a ; -r_4$ is applied, sending one spike to neurons $\sigma_7$ and $\sigma_{l_2}^{(2)}$, respectively, and removing rule $r_4$. At step $t + 4$, $\sigma_7$ applies $a \rightarrow \lambda$, forgetting the one spike from neuron $\sigma_{l_1}^{(1)}$. In addition, $r_8 : a^5 \rightarrow a^4$ is applied in neuron $\sigma_{l_2}^{(2)}$, sending four spikes to $\sigma_7$. At step $t + 5$, $\sigma_k$ contains four spikes and starts simulating $l_k$.

Table 1 lists five variants of SNP systems and the number of neurons they require to construct the ADD module. From Table 1, it can be seen that DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR require 7, 8, 11, 9, and 8 neurons, respectively, while RDGRSNP
requires 6. Therefore, the proposed RDGRSNP system in this paper requires the least number of neurons.

Table 1. The comparison of the number of neurons required to construct the ADD module.

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>7</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>8</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>11</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>9</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>8</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>6</td>
</tr>
</tbody>
</table>

**SUB module:** The SUB module is used to simulate instruction \( l_i : (\text{SUB}(r), l_j, l_k) \), as shown in Figure 4. This module consists of six neurons and twelve rules.

![Figure 4. SUB module.](image)

Suppose that neuron \( \sigma_l \) contains four spikes at step \( t \) and starts simulating instruction \( l_i : (\text{SUB}(r), l_j, l_k) \). Neuron \( \sigma_l \) applies the spiking rule with rules dynamic generation and removal \( r_1 : a^4 / a \rightarrow a; r_2 \), sends one spike to \( \sigma_{l_1}^{(1)} \), \( \sigma_{l_2}^{(2)} \), and \( \sigma_r \), and adds \( r_2 \) to the rule set.

At step \( t + 1 \), \( \sigma_l \) remains with three spikes with rules \( r_1, r_2, \) and \( r_8 \), so \( \sigma_l \) applies the spiking rule with rules dynamic generation and removal \( r_2 : a^3 \rightarrow a^2 ; r_2 \), sends two spikes to the postsynaptic neuron, and removes the rule \( r_2 \). In \( \sigma_{l_1}^{(1)} \) and \( \sigma_{l_2}^{(2)} \), the forgetting rule is applied, forgetting one spike. In addition, different rules are applied in neuron \( \sigma_r \) depending on whether the initial state contains spikes or not in the following two cases:

1. If \( \sigma_r \) contains \( 4n(n \geq 1) \) spikes, then at step \( t + 1 \), \( \sigma_r \) applies \( r_3 : a(a^4)^+ / a \rightarrow a; r_4 \), consumes one spike, and sends it to \( \sigma_{l_1}^{(1)}, \sigma_{l_2}^{(2)} \), and \( \sigma_r \), and adds \( r_2 \) to the rule set. Additionally, neuron \( \sigma_l \) contains \( 4n + 2 \) spikes with rules \( r_3, r_4 \) and \( r_6 \). Thus, the spiking rule with rules dynamic generation and removal \( r_4 : a^2(a^4)^+ / a^2 \rightarrow a^2 ; r_5, r_4 \) is applied, sending two spikes to \( \sigma_{l_1}^{(1)} \) and \( \sigma_{l_2}^{(2)} \), adding rule \( r_5 \), and removing rule \( r_4 \). At step \( t + 3 \), \( \sigma_r \) applies \( r_5 : (a^4)^+ / a^4 \rightarrow \lambda; r_5 \), forgetting four spikes. Thus, at step \( t + 4 \), \( \sigma_r \) has \( 4n - 4 \) spikes, corresponding to the number in register \( r \) subtracted by one. Additionally, at step \( t + 3 \), \( \sigma_{l_1}^{(1)} \) applies \( r_{11} : a^5 \rightarrow a^4 \) to send four spikes to \( \sigma_{l_1}^{(1)} \). In neuron \( \sigma_{l_2}^{(2)} \), the forgetting rule \( r_{12} : a^5 \rightarrow \lambda \) is applied. At step \( t + 4 \), \( \sigma_{l_1}^{(1)} \) contains four spikes, simulating instruction \( l_j \).}

2. If neuron \( \sigma_r \) does not contain spikes, then at step \( t + 1 \), the spiking rule with rules dynamic generation and removal \( r_6 : a \rightarrow a ; r_7 \) is applied, sending one spike to \( \sigma_{l_1}^{(1)} \).
and \( \sigma_{i[2]} \) and adding \( r_7 \). At step \( t + 2 \), \( \sigma_t \) has \( r_3, r_6 \), and \( r_7 \) with two spikes, so the spiking rule with rules dynamic generation and removal \( r_7 : a^2 \rightarrow a; \neg r_7 \) is applied, removing rule \( r_7 \). Thus, at step \( t + 3 \), the auxiliary neuron \( \sigma_{i[1]} \) applies rule \( r_{10} : a^4 \rightarrow \lambda \), while \( \sigma_{i[2]} \) applies rule \( r_6 : a^4 \rightarrow a^4 \), sending spikes to \( \sigma_l \). At step \( t + 4 \), \( \sigma_l \) contains four spikes, simulating instruction \( l_k \).

Table 2 lists five variants of SNP systems and the number of neurons they require to construct the SUB module. From Table 2, it can be seen that DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR require 7, 10, 15, 8, and 8 neurons, respectively, while RDGRSNP requires 6. Therefore, the proposed RDGRSNP system in this paper requires the least number of neurons.

### Table 2. The comparison of the number of neurons required to construct the SUB module.

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>7</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>10</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>15</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>8</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>8</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>6</td>
</tr>
</tbody>
</table>

**FIN module**: The FIN module is used to simulate the halt instruction \( l_h : HALT \) and output the numbers generated by system \( \Pi \), as shown in Figure 5. The generated number \( n \) is represented by the time interval between the first two spikes sent to the environment \( env \) by the output neuron. The FIN module constructed in this paper consists of three neurons and six rules.

![Figure 5. FIN module.](image)

Suppose that at step \( t \), \( \sigma_l \) has four spikes, starts simulating instruction \( l_h : HALT \) and outputting the computation results. Neuron \( \sigma_{i_h} \) applies the spiking rule with rules dynamic generation and removal \( r_1 : a^4 / a^2 \rightarrow a; r_2 \), sends one spike to \( \sigma_1 \) and \( \sigma_{out} \), and adds \( r_2 \) to it. Thus, at step \( t + 1 \), neuron \( \sigma_{i_h} \) applies \( r_2 : a^2 \rightarrow a; r_2 \), sends one spike to \( \sigma_1 \) and \( \sigma_{out} \), and removes \( r_2 \). Neuron \( \sigma_1 \) applies the spiking rule with rules dynamic generation and removal \( r_3 : a(a^4)^t / a^5 \rightarrow a; r_4, r_5, r_3 \), consuming five spikes, sending one spike to the output neuron \( \sigma_{out} \), adding rules \( r_4 \) and \( r_5 \), and removing rule \( r_3 \). At step \( t + 2 \), \( \sigma_{out} \) applies \( r_5 : a^3 \rightarrow a; r_5 \) sending the first spike to \( env \) and adding \( r_5 \). So, starting from step \( t + 2 \), neuron \( \sigma_1 \) will forget four spikes at each step until step \( t + n + 1 \), where \( n \geq 1 \).

At step \( t + n + 1 \), \( \sigma_1 \) has only one spike with rules \( r_4 \) and \( r_5 \). Therefore, it applies the spiking rule with rules dynamic generation and removal \( r_5 : a \rightarrow a; r_3, r_4, r_5, r_3 \), sends one spike to \( \sigma_{out} \), adds \( r_3 \), and removes rules \( r_4 \) and \( r_5 \). At step \( t + n + 2 \), \( \sigma_{out} \) applies \( r_5 : a \rightarrow a; r_3, r_5 \), to send the second spike to \( env \). Thus, the time interval between the first two spikes sent by \( \sigma_{out} \) to the environment is \((t + n + 2) - (t + 1) = n\), i.e., the number stored in register 1 when the computation stops.

Table 3 lists five variants of SNP systems and the number of neurons they require to construct the FIN module. From Table 3, it can be seen that DSNP, PASNP, PSNRSP,
MPAIRSNP, and SNP-IR require 4, 9, 8, and 5 neurons, respectively, while RDGRSNP requires 3. Therefore, the proposed RDGRSNP system in this paper requires the least number of neurons.

Table 3. The comparison of the number of neurons required to construct the FIN module.

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>4</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>9</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>8</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>8</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>5</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>3</td>
</tr>
</tbody>
</table>

**Theorem 2.** $N_{acc} RDGRSNP(Π) = NRE$

**Proof.** Similarly, by Turing Church’s theorem, we have $N_{acc} RDGRSNP(Π) \subseteq NRE$, so we only need to prove $NRE \subseteq N_{acc} RDGRSNP(Π)$. For $NRE \subseteq N_{acc}(Π)$, we simulate the register machine to prove it by following the process shown in Figure 6. □

![Flowchart of the RDGRSNP system simulating the register machine in number-accepting mode.](image)

In the initial state, neuron $σ_1$ contains $4n$ spikes, corresponding to the number $n$ to be accepted in the register machine, neuron $σ_{nR}$ contains four spikes for triggering the simulation of computation, and all the remaining neurons are empty. The simulation of computation follows the simulation of instructions in the register machine until neuron $σ_{nR}$ receives four spikes. When neuron $σ_{nR}$ receives four spikes, it means that the computation process in the register machine is simulated successfully, the computation stops, and the number is accepted.

First, in the number-accepting mode, we need the INPUT module to read the number to be accepted. The number $n$ to be accepted is represented by the time interval between the first two spikes entered, i.e., the spike train entered is: $10^n−11$.

**INPUT module:** The INPUT module is used to read the number to be accepted, as shown in Figure 7. This module consists of nine neurons and seven rules.

Suppose that at step 0, $σ_{in}$ receives one spike, so at step 1, $σ_{in}$ applies $r_3: a \rightarrow a$ and sends one spike to auxiliary neurons $σ_{in1}, σ_{in2}, σ_{in3},$ and $σ_{in4}$, respectively. Starting from step 2, auxiliary neurons $σ_{in1}, σ_{in2}, σ_{in3},$ and $σ_{in4}$ will send spikes in a cycle and send one spike to $σ_{ins}$ and $σ_{ins'}$, respectively. For neuron $σ_{ins}$, it applies $r_4: a^k \rightarrow a^k$ to send...
four spikes to $\sigma_1$; for neuron $\sigma_{in}$, it applies rule $r_5 : a^4 \rightarrow \lambda$ to forget the received four spikes. The above process will continue until the input neuron $\sigma_{in}$ receives another spike.

![INPUT module](image)

**Figure 7.** INPUT module.

At step $n$, $\sigma_{in}$ receives another spike, so at step $n+1$, $\sigma_{in}$ applies $r_5 : a \rightarrow a$ again, sends one spike to $\sigma_{in1}$, $\sigma_{in2}$, $\sigma_{in3}$, and $\sigma_{in4}$. At step $n+2$, $\sigma_{in1}$, $\sigma_{in2}$, $\sigma_{in3}$, and $\sigma_{in4}$ all apply the spiking rule with rules dynamic generation and removal $r_1 : a^2 \rightarrow a^2$, $r_2 : \lambda \rightarrow -r_1$, send two spikes to neurons $\sigma_{in5}$ and $\sigma_{in6}$, remove rule $r_1$, and add rule $r_2$. Thus, at step $n+3$, these four auxiliary neurons apply rule $r_2 : a^2 \rightarrow \lambda$; $r_1, -r_2$ to forget two spikes. At step $n+3$, both neurons $\sigma_{in5}$ and $\sigma_{in6}$ receive eight spikes. Neuron $\sigma_{in5}$ applies rule $r_7 : a^8 \rightarrow \lambda$ to forget the spikes in it. Neuron $\sigma_1$ receives four spikes at step $n+2$, so it contains $4n$ spikes. In $\sigma_{in6}$, rule $r_6 : a^8 \rightarrow a^4$ is applied to send four spikes to $\sigma_0$. Then, neuron $\sigma_0$ contains four spikes, simulating the starting instruction $l_0$.

Table 4 lists five variants of SNP systems and the number of neurons they require to construct the INPUT module. From Table 4, it can be seen that DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR require 6, 6, 10, 6, and 9 neurons, respectively, while RDGRSNP requires 9. However, this is acceptable because the INPUT module will be used only once in the number-generating mode and will not have a great impact on the overall number of neurons.

**Table 4.** The comparison of the number of neurons required to construct the INPUT module.

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>6</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>6</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>10</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>6</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>6</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>9</td>
</tr>
</tbody>
</table>

In the number-generating mode, the SUB instruction in $M$ does not change, so we can continue to use the SUB module in Theorem 1 above. Since the system stop means that the input numbers are accepted, an additional FIN module is no longer needed to output the computation results. However, in the number-accepting mode, the ADD instruction in $M$ is no longer in the non-deterministic form of $l_i : (ADD(r), l_j, l_k)$, but in the deterministic form of $l_i : (ADD(r), l_j)$. Therefore, this paper provides a deterministic ADD module, as shown in Figure 8. This module consists of only three neurons and three rules.
In this deterministic ADD module, $\sigma_l$ sends spikes to $\sigma_r$, simulating the operation of adding one to the number of register $r$ and, at the same time, sends spikes to $\sigma_j$, causing instruction $l_j$ to start being simulated.

Table 5 lists five variants of SNP systems and the number of neurons they require to construct the deterministic ADD module. From Table 5, it can be seen that DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR require 4, 3, 5, 3, and 3 neurons, respectively, while RDGRSNP requires 3. Therefore, PASNP, MPAIRSNP, SNP-IR and the proposed RDGRSNP system in this paper require the least number of neurons.

Table 5. The comparison of the number of neurons required to construct the deterministic ADD module.

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>4</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>3</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>5</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>3</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>3</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>3</td>
</tr>
</tbody>
</table>


This section presents a small universal RDGRSNP system used to simulate function computation and demonstrates that the RDGRSNP system we proposed requires fewer neurons by comparing it with five variants of SNP systems.

Theorem 3. A small universal RDGRSNP system using 68 neurons can implement function simulation.

Regarding Theorem 3, we continue to prove it by simulating a register machine. In $M_u [41]$, for simulating the function computation shown in Figure 9, there always exists a recursive function $g$ such that $\theta_i(y) = M_u(g(x), y)$ holds for any $\theta_i(y)$, where $\theta_i(y)$ is any one function of the fixed enumeration $(\theta_1, \theta_2, \ldots)$ of all unary partial recursive functions, $M_u$ denotes a universal register machine, and $g(x)$ and $y$ are two parameters with $x, y \in N$ stored in registers 1 and 2, respectively. The register machine $M_u$ for the simulating function computation stops when the execution reaches $l_h : \text{HALT}$. The result of the computation is stored in register 0.
\[ l_k : (SUB(1), l_1, l_2), \quad l_1 : (ADD(7), l_3), \]
\[ l_2 : (ADD(6), l_1), \quad l_1 : (SUB(5), l_1, l_4), \]
\[ l_3 : (SUB(6), l_3, l_6), \quad l_2 : (ADD(5), l_6), \]
\[ l_4 : (SUB(7), l_7, l_6), \quad l_7 : (ADD(1), l_4), \]
\[ l_5 : (SUB(6), l_6, l_9), \quad l_9 : (ADD(6), l_{10}), \]
\[ l_{10} : (SUB(4), l_{11}), \quad l_{11} : (SUB(5), l_{12}, l_{13}), \]
\[ l_{12} : (SUB(5), l_{14}, l_{15}), \quad l_{13} : (SUB(2), l_{16}, l_{19}), \]
\[ l_{14} : (SUB(5), l_{16}, l_{17}), \quad l_{15} : (SUB(3), l_{18}, l_{19}), \]
\[ l_{16} : (ADD(4), l_{11}), \quad l_{17} : (ADD(2), l_{21}), \]
\[ l_{18} : (SUB(4), l_6, l_9'), \quad l_{19} : (SUB(0), l_0, l_{18}), \]
\[ l_{20} : (ADD(0), l_0), \quad l_{21} : (ADD(3), l_9'), \]
\[ l_{22} : (SUB(0), l_{23}, l_9'), \quad l_{23} : (ADD(8), l_{22}), \]
\[ l_9' : HALT \]

Figure 9. Register machine \( M_d \).

**Proof:** For Theorem 3, we simulate the register machine to prove it by following the process shown in Figure 10.

![Flowchart of a small universal RDGRSNP system simulating a register machine](image)

**Figure 10.** The flowchart of a small universal RDGRSNP system simulating a register machine.

In the initial state, neuron \( \sigma_1 \) contains \( 4g(x) \) spikes, corresponding to parameter \( g(x) \) in register 1, neuron \( \sigma_2 \) contains \( 4y \) spikes, corresponding to parameter \( y \) in register 2, neuron \( \sigma_0 \) contains four spikes for triggering the simulation of computation, and all the remaining neurons are empty. The simulation of computation follows the simulation of instructions in the register machine until neuron \( \sigma_0 \) receives four spikes. When neuron \( \sigma_0 \) receives four spikes, it means that the computation process of the register machine is successfully simulated and the computation stops. Meanwhile, the FIN module is triggered and used to output the computation result.

In \( M_d \), the SUB instruction in the form of \( l_i : (SUB(r), l_j, l_k) \), the halt instruction \( l_h : HALT \), and the deterministic ADD instruction in the form of \( l_i : (ADD(r), l_j) \) are included. Thus, we can continue to use the SUB, FIN, and deterministic ADD module, as proposed in Theorems 1 and 2 above. In addition, since two registers are needed to store parameters \( g(x) \) and \( y \), respectively, this paper makes a little change to the INPUT module in Theorem 2 above, as shown in Figure 11. We still use the time interval between two spikes to represent the input number, so the input sequence is \( 10^{g(x)-1}10^{y-1}1 \).
Suppose that at step 0, \( \sigma_{in} \) receives the first spike, so at step 1, \( \sigma_{in} \) applies \( r_3 : a \rightarrow a \) and sends one spike to the four auxiliary neurons \( \sigma_{in1}, \sigma_{in2}, \sigma_{in3}, \) and \( \sigma_{in4} \). Thus, from step 2, these four auxiliary neurons send spikes in a cycle by applying rule \( r_3 : a \rightarrow a \), and all send one spike to \( \sigma_{in5}, \sigma_{in6}, \) and \( \sigma_{in7} \). For neuron \( \sigma_{in5} \), starting at step 3, it continuously applies \( r_5 : a^3 \rightarrow a^4 \) to send four spikes to \( \sigma_1 \), by which the first input number is stored in \( \sigma_1 \), while for neurons \( \sigma_{in6}, \sigma_{in7} \), they continuously apply rule \( r_6 : a^4 \rightarrow \lambda \) to forget four spikes. The process continues until the input neuron receives the second spike.

At step \( g(x) \), \( \sigma_{in} \) receives the second spike, so at step \( g(x) + 1 \), \( \sigma_{in} \) applies \( r_3 : a \rightarrow a \) again, sends one spike to neurons \( \sigma_{in1}, \sigma_{in2}, \sigma_{in3}, \) and \( \sigma_{in4} \). Thus, starting at step \( g(x) + 2 \), these four neurons apply rule \( r_4 : a^2 \rightarrow a^3 \) to send two spikes in a cycle, while sending two spikes to their postsynaptic neurons \( \sigma_{in5}, \sigma_{in6}, \) and \( \sigma_{in7} \). Similarly, for neuron \( \sigma_{in6} \), starting at step \( g(x) + 3 \), it applies rule \( r_7 : a^3 \rightarrow a^4 \) continuously to send four spikes to the neuron \( \sigma_2 \), by which the second input number is stored in \( \sigma_2 \). For neurons \( \sigma_{in5} \) and \( \sigma_{in7} \), they apply rule \( r_8 : a^4 \rightarrow \lambda \) to forget the spikes. The process continues until \( \sigma_{in} \) receives the third spike. In addition, \( \sigma_1 \) receives four spikes sent from \( \sigma_{in5} \) at step \( g(x) + 2 \). Therefore, \( \sigma_1 \) contains \( 4g(x) \) spikes, which corresponds to the number \( g(x) \) in register 1.

At step \( g(x) + y \), \( \sigma_{in} \) receives the third spike, so at step \( g(x) + y + 1 \), \( \sigma_{in} \) applies \( r_3 : a \rightarrow a \) again and sends one spike to \( \sigma_{in1}, \sigma_{in2}, \sigma_{in3}, \) and \( \sigma_{in4} \). At step \( g(x) + y + 2 \), neurons \( \sigma_{in1}, \sigma_{in2}, \sigma_{in3}, \) and \( \sigma_{in4} \) all contain three spikes with rules \( r_1, r_2, \) and \( r_d \), so they all apply the spiking rule with rules dynamic generation and removal \( r_1 : a^3 \rightarrow a^3, r_2, r_d \), and rules dynamic generation and removal \( r_2 : a^3 \rightarrow a^2, r_2, \) and rules dynamic generation and removal \( r_1 \) again. Meanwhile, both neurons \( \sigma_{in5} \) and \( \sigma_{in6} \) contain 12 spikes and apply rule \( r_{10} : a^{12} \rightarrow \lambda \) to forget these 12 spikes. Neuron \( \sigma_{in7} \) applies rule \( r_9 : a^{12} \rightarrow a^4 \) to send four spikes to \( \sigma_{l_0} \). In addition, \( \sigma_2 \) receives four spikes sent by neuron \( \sigma_{in6} \) at step \( g(x) + y + 2 \), so it has \( 4y \) spikes, which corresponds to the number \( y \) in register 2. At step \( g(x) + y + 4 \), \( \sigma_{l_0} \) contains four spikes, simulating the starting instruction \( l_0 \).

Table 6 lists five variants of SNP systems and the number of neurons they need to build the INPUT module in the register machine. From Table 6, it can be seen that DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR require 9, 9, 13, 9, and 9 neurons, respectively, while RDGRSNP requires 11. However, this is acceptable because the INPUT module will be used only once during the simulation of function calculation and will not have a great impact on the overall number of neurons.
Table 6. The comparison of the number of neurons required to construct the INPUT module in register machine \( M_u \).

<table>
<thead>
<tr>
<th>Variants of SNP Systems</th>
<th>Number of Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNP [38]</td>
<td>9</td>
</tr>
<tr>
<td>PASNP [39]</td>
<td>9</td>
</tr>
<tr>
<td>PSNRSP [40]</td>
<td>13</td>
</tr>
<tr>
<td>MPAIRSNP [41]</td>
<td>9</td>
</tr>
<tr>
<td>SNP-IR [42]</td>
<td>9</td>
</tr>
<tr>
<td>RDGRSNP</td>
<td>11</td>
</tr>
</tbody>
</table>

According to the INPUT, deterministic ADD, SUB, and FIN modules described above, a small universal RDGRSNP system for simulating function \( \theta_x(y) = M_u(g(x), y) \) consists of 71 neurons, as follows:

1. Neurons used to start simulating instructions: 25
2. Neurons used to simulate registers: 9
3. Auxiliary neurons in the ADD module: 0
4. Auxiliary neurons in the SUB module: \( 2 \times 14 = 28 \)
5. The input and auxiliary neurons in the INPUT module: 8
6. The output neuron in the FIN module: 1

We can also further reduce the number of neurons by compound connections between modules, and the following is the specific structure of compound connections between modules.

Figure 12 represents the composite connection of a set of consecutive ADD instructions \( l_i : (ADD(r', l_i^g), l_i^g) \) and \( l_k : (ADD(r'', l_j), l_j) \). We put \( \sigma_{r'} \) from the former ADD module into the latter module, removing \( \sigma_{l_i^g} \).

We can also further reduce the number of neurons by compound connections between modules, and the following is the specific structure of compound connections between modules.

Figure 12 represents the composite connection of a set of consecutive ADD instructions \( l_i : (ADD(r', l_i^g), l_i^g) \) and \( l_k : (ADD(r'', l_j), l_j) \). We put \( \sigma_{r'} \) from the former ADD module into the latter module, removing \( \sigma_{l_i^g} \).

By the compound connection between the modules mentioned above, we remove three neurons \( \sigma_{l_{21}}, \sigma_{l_{6}}, \) and \( \sigma_{l_{10}} \), so that the number of neurons required to build a small universal RDGRSNP system for simulating function \( \theta_x(y) = M_u(g(x), y) \) is reduced from 71 to 68.

Table 7 lists several variants of SNP systems and the number of neurons they require to construct a small universal SNP system for simulating function computation. From the table, it can be seen that five variants of SNP systems, DSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR, require 81, 121, 151, 95, and 100 neurons, respectively, while the RDGRSNP system proposed in this paper requires only 68.
In the construction of SNP systems, the number of neurons is generally used to measure the computational resources required to build a system. A smaller number of neurons means fewer computational resources are required to build a system; conversely, a larger number of neurons means more resources are required to build a system. Therefore, the smaller the number of neurons needed to construct the SNP system, the better. As we can see in Table 7, our proposed RDGRSNP systems require fewer neurons to build a small universal SNP system for simulating function computation. This means that a RDGRSNP system needs only fewer computational resources compared to other systems. Thus, our proposed RDGRSNP systems have an advantage.

5. Conclusions

In conventional SNP systems, the rules contained in neurons do not change during the computation process. However, biochemical reactions in biological neurons tend to be different depending on factors such as the substances in the neuron. Considering this motivation, RDGRSNP systems are proposed in this paper. In RDGRSNP systems, neurons apply rules to update the rule set. In Section 2, we give the definition of RDGRSNP systems and illustrate how RDGRSNP systems work with an illustrative example.

In Section 3, we demonstrate the computational power of RDGRSNP systems by simulating the register machines. Specifically, we demonstrate that RDGRSNP systems are Turing universal when used as a number-generating device and a number-accepting device. Subsequently, in Section 4, we construct a small universal RDGRSNP system by using 68 neurons. By comparing with five variants of SNP systems, DSNP, Asynchronous RSSNP, PASNP, PSNRSP, MPAIRSNP, and SNP-IR, it is demonstrated that the RDGRSNP system proposed in this paper requires only fewer resources to construct a small universal RDGRSNP system for simulating function computation.

Our future research will focus on the following areas. Although the computational power of RDGRSNP systems has been demonstrated, the potential of RDGRSNP systems...
goes far beyond that. SNP systems have shown excellent capabilities in solving NP problems, and we have demonstrated the advantage of RDGRSNP systems compared with other variants of SNP systems. Therefore, we believe that RDGRSNP systems can perform better in solving NP problems.

RDGRSNP systems proposed in this paper work in sequential mode. However, there are other modes of operation such as asynchronous mode that have not been discussed. In asynchronous mode, the neurons in the system can choose to apply or not to apply the rules at each step. Of course, these rules also need to satisfy control conditions to be applied. Therefore, neurons have more autonomy in asynchronous mode. Many variants of SNP systems have been discussed working in asynchronous mode. An exploration of RDGRSNP systems working in asynchronous mode is also a future research direction for this paper.

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

Funding: This work was supported by National Natural Science Foundation of China (Nos. 61806114, 61876101) and China Postdoctoral Science Foundation (Nos. 2018M642695, 2019T120607).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References
41. Liu, Y.; Zhao, Y. Spiking Neural P Systems with Membrane Potentials, Inhibitory Rules, and Anti-Spikes. *Entropy* 2022, 24, 834. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.