Spiking Neural P Systems With Enzymes

Xiang Tian, Xiyu Liu, Qianqian Ren, and Yuzhen Zhao

***Abstract*— The neurotransmitter is a chemical substance that transmits information between neurons. Its metabolic process includes four links: synthesis, storage, release and inactivation. As one of the important chemical components of neurotransmitters, acetylcholine is synthesized under the catalysis of acetylcholine coenzyme A and choline acetylase. Inspired by the biological fact that enzymes exist in neurons and that enzymes are involved in neurotransmitter synthesis, we propose spiking neural P systems with enzymes (SNPE). Different from the previous spiking neural P systems and their variants, each neuron of SNPE contains two classes of objects, and each spiking rule has the participation of enzymes. In addition, the number of spikes and enzymes in a neuron can also serve as a consumption condition for controlling whether a reaction (rule execution) occurs. When the number of enzymes meets the requirements of a specific biochemical reaction, the number of occurrences of the reaction can also be controlled. As number generation and acceptance devices, the proposed SNPE systems are proved to be Turing universal. In addition, 61 neurons are used to construct an SNPE system that realizes function computation, which proves the Turing universality in this mode. Finally, we also explore using a uniform SNPE model to solve the subset sum problem and compare it with the standard SN P and its several variants.**

***Index Terms*— Membrane computing, spiking neural P systems, biocomputing, enzymes, subset sum, NP-complete.**

I. Introduction

MEMBRANE computing is a cell-level biological abstraction between DNA molecules and neural networks. It reflects the biochemical mechanism of information transmission, cooperative transport and possible structural evolution in living cells or between cells. The concept and system theory of membrane computing was first proposed by Păun [1]. Therefore, the membrane system is also referred to as the P system [2] for short. The membrane computing models have attracted a large number of scholars to conduct extensive and in-depth research due to their outstanding characteristics such as distribution, parallelism and non-determinism. According to different topological structures and biological principles, generally speaking, P systems mainly involve three types in topology [3]: cell-like P systems (hierarchical topological features), tissue-like P systems (network-like topological features), and neural-like P systems (topological features appear as directed graphs).

In the nervous system of living organisms, nerve cells (neurons) exchange information by transmitting spikes through synapses. Inspired by this biological fact, spiking neural P systems (SN P) were first proposed [4] with their concise expression and efficient structures. Since then, further inspired by various biological facts such as the biochemical reactions and functional structure of the biological nervous system, various variants based on the original SN P systems framework are constantly being developed. SN P has become the most promising membrane computing model. Considering the excitatory and inhibitory effects of astrocytes on synapses, Păun [5] and Pan *et al.* [6] studied the SN P systems with astrocytes. On this basis, Aman and Ciobanu [7] further studied the SN P systems with astrocytes that can produce calcium ions. Pan and Păun [8] abstracted the concept of anti-pulse from the biological phenomenon of inhibitory pulse, and proposed the SN P systems with anti-spikes. By introducing white hole rules, SN P systems with white hole neurons [9] were constructively proposed. The SN P systems with polarizations were fully discussed and studied [10], [11]. Peng *et al.* [12], Huang *et al.* [13], and Lv *et al.* [3] discussed the SN P systems with multiple channels rules. Cavaliere *et al.* [14] proposed the SN P systems with non-synchronous (i.e., asynchronous) rules for the first time and proved its equivalence with Turing machines. Based on this, Pan *et al.* [15] further proposed a new working mode called limited asynchronous SN P systems. Peng *et al.* [16] studied fuzzy reasoning SN P and applied it in the field of fault diagnosis. In terms of the execution of rules, the existing mainstream modes are divided into the following three categories: sequential mode [4], exhaustive mode [17], [18], and flat maximal parallel mode [19], [20]. Wu *et al.* [21] first introduced the flat maximally parallel mode into the SN P systems.

Zeng *et al.* [22] first introduced the concept and idea of threshold to SN P systems. Inspired by the phenomenon of synchronized oscillations of neurons in cat's visual cor-

Manuscript received 29 December 2021; revised 9 May 2022 and 24 June 2022; accepted 13 August 2022. Date of publication 18 August 2022; date of current version 3 October 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61876101, Grant 61806114, Grant 61802234, and Grant 62172226; in part by the Humanities and Social Sciences Youth Foundation, Ministry of Education of the People's Republic of China, under Grant 19YJCZH244; in part by the Natural Science Foundation of Shandong Province under Grant ZR2019QF007; in part by the China Postdoctoral Science Foundation Funded Project under Grant 2017M612339 and Grant 2018M642695; and in part by the China Postdoctoral Special Funding Project under Grant 2019T120607. *(Corresponding authors: Xiyu Liu; Yuzhen Zhao.)*

Xiang Tian is with the Business School, Shandong Normal University, Jinan 250358, China (e-mail: tianxiang08@163.com).

Xiyu Liu and Yuzhen Zhao are with the Business School, Academy of Management Science, Shandong Normal University, Jinan 250358, China (e-mail: xyliu@sdnu.edu.cn; zhaoyuzhen@sdnu.edu.cn).

Qianqian Ren is with the School of Computer Science, Qufu Normal University, Rizhao 276826, China.

Digital Object Identifier 10.1109/TNB.2022.3199767

tex, Peng *et al.* [23] further conceived the dynamic threshold neural P systems (DTN P). Subsequently, Huang *et al.* [24] discussed the computing power of the DTN P used to generate string languages. Considering that there may be multiple synapses between biological neurons to realize connections, Wang *et al.* [25] and Pan *et al.* [26] introduced the idea of weighting to the synaptic connections between neurons. Inspired by the biological characteristics of neuroplasticity, Cabarle *et al.* [27] defined the generation and deletion of synapses between neurons as a kind of structural plasticity. Cruz *et al.* [28] further explored homogeneous SN P with structural plasticity, where each neuron has the same set of rules. Similarly, Wang *et al.* [29] designed self-organizing SN P systems to realize the dynamic connection of synapses between neurons in the computing process. Based on the neural model of the mammalian visual cortex, a novel coupled neural P system was proposed [30], in which each coupled neuron is composed of a receptive field, a modulation module, and an output module. Inspired by numerical P systems, Wu *et al.* [31] proposed numerical SN P systems. Peng *et al.* [32] cleverly introduced the non-linear activation function commonly used in deep neural networks as a non-linear spiking rule, and extended the spike count unit from integer level to real number level. Considering the biological facts of inhibitory synapses, Peng *et al.* [33] introduced inhibitory rules. Neary [34] and Zhang *et al.* [35] studied small universal SN P systems using extension rules.

In addition, a large number of scholars have carried out research and discussion on the combination of the above-mentioned variants. For example, Ren *et al.* [36] combined weighting ideas with anti-spikes. Wu *et al.* [37] merged polarization and asynchronous rules. Song *et al.* [38] discussed the Turing universality of asynchronous rules with multiple channels. Yang *et al.* [39] combined anti-spikes and structural plasticity and proposed extended SN P systems. Pan *et al.* [40] incorporated the cell-like structure into SN P systems, and also further introduced request rules, which allow the skin membrane to obtain pulse information from the outside. Song *et al.* [41] explored integrating anti-spikes into the asynchronous SN P systems. SN P systems also showed some groundbreaking results in application. SN P systems with fuzzy reasoning [16] were successfully applied in the field of fault diagnosis. SN P systems with learning functions were proposed and successfully used to solve the problem of letter recognition [42]. SN P systems were also well applied in decoder design [43], classification tasks [44], and workflow modeling [45].

Neurotransmitters are chemicals that help signals pass from one neuron to another. It combines with specific receptors on the postsynaptic cell membrane to affect the membrane potential of postsynaptic neurons or cause the physiological effects of effector cells to complete synaptic information transmission. The metabolic process of neurotransmitter includes four links: synthesis, storage, release and inactivation. As one of the important chemical components of neurotransmitters, acetylcholine is synthesized under the catalysis of acetylcholine coenzyme A and choline acetylase, and then transferred to vesicles for storage. When the nerve impulse reaches the nerve terminal, the vesicle membrane and the presynaptic membrane fuse to release acetylcholine into the synaptic cleft. At the same time, acetylcholine is hydrolyzed into choline and acetic acid by cholinesterase (ChE) in the nerve endings and inactivated. Part of the choline is once again taken up by the cholinergic nerve endings and participates in the synthesis of new acetylcholine.

This work is inspired by the above biological principles, and proposes new spiking neural P systems with enzymes (SNPE). Compared with the standard SN P and its variants, SNPE has made improvements in terms of objects, rules, and system operation. As a function computing device, the proposed SNPE is compared with 6 SN P variants published in the past 3 years (2019∼2021). In addition, we also explore the computational power of SNPE systems to uniformly solve the subset sum problem attributed to NP-complete. The detailed inspiration, motivation, innovations and contributions of this work will be given in the next section.

The logical structure of this work is arranged below. Section II describes the relevant basic knowledge. Motivations, the formal definition of the SNPE system and an example are presented in Section III. Section IV gives two proofs of Turing universality of SNPE systems as number generation and acceptance devices, respectively. The SNPE system as a functional computing device and its computing power are shown in Section V. Section VI demonstrates the computational power of the SNPE system to uniformly solve the subset sum problem attributed to NP-complete. A summary and outlook are in Section VII.

II. Preliminary Knowledge

In this section, we briefly review some necessary knowledge about formal languages and automata theory. At the same time, the commonly used notations and their meanings throughout this work are given.

Suppose that V is an alphabet, a set with non-emptiness and finiteness. A finite sequence of elements (also called characters) in alphabet V formed one after another in a certain order is called a string. The empty string λ does not contain any characters. The meaning of V^* is similar to the Kleene closure, and V^+ is similar to the positive closure. The union of V^+ and λ is equivalent to V^*. For a detailed introduction to formal languages, readers can also refer to [2] and [46].

The formal definition of a register machine is denoted by the tuple $M = (m, H, l_0, l_h, I)$, where, m denotes the number of registers. H represents a set of instruction labels. l_0 and l_h mean the starting label and the halting label, respectively. I is the set of all possible instructions. Every label in H corresponds to an instruction in I one-to-one. The instructions in I are divided into three types, that is, the addition instruction $l_i : (ADD(r), l_j, l_k)$, the subtraction instruction l_i: $(SUB(r), l_j, l_k)$ and the termination instruction $l_h : HALT$. For a further detailed description of register machine, please refer to [47].

The computing power of a new computing model is usually verified by realizing the simulation of the register machine.

It is usually considered to verify separately from the perspective of the two modes. One is the generation mode, and the working principle of this mode is as follows. Initially, the stored value in each register is empty. The calculation of the system starts at l_0 and ends at l_h. At this time, the value stored in register 1 is the number computed (or generated) by M. Use $N_{gen}(M)$ to represent the set of numbers generated by the register machine M in this mode.

The other is acceptance mode which is different from the generation mode in two main points. On the one hand, in this mode, a certain register needs to be designated as an input unit initially, and the storage value of the remaining registers is empty. On the other hand, the addition instruction in the acceptance mode is deterministic, abbreviated as $l_i : (ADD(r), l_j)$. Use $N_{acc}(M)$ to represent the set of numbers that can be accepted by the register machine in this mode.

III. Motivations and the Proposed SNPE Systems

The research motivation and innovative mechanism of the proposed SN P systems with enzymes (SNPE) are inspired by the following biological principles. (i) The firing of information in neurons requires the participation of enzymes. (ii) Enzymes are only present in the current neuron, and do not pass and communicate between neurons as information. (iii) Usually, enzymes consumed by participating in the reaction can be synthesized automatically to ensure the sustainability of subsequent reactions in that neuron. Based on the inspiration of the above biological principles, the innovations and contributions of this work can be listed below.

i. The concept of the enzyme is introduced into the SN P system for the first time. The proposed SNPE systems will contain double objects, i.e., spike and enzyme, where the enzyme is only present in the current neuron.
ii. The number of enzymes and spikes together constitute the consumption condition of the rules, which not only determines which rule can be executed, but also determines the number of times a rule can be executed in parallel.
iii. As a function computing device, comparisons with several SN P variants proposed in the last 3 years are made. The comparison results show that the number of neurons used by the SNPE model and the maximum number of rules used in each neuron are both the smallest.
iv. Finally, we also explore using a uniform SNPE model to solve the subset sum problem and compare it with the standard SN P and its latest several variants. The enzyme, as a controller of whether the rules are executed sustainably, is more flexible and prominent here. That is, when the enzymes are not renewable, the execution of the rules is no longer sustainable. The comparison results show that the proposed SNPE model has a more compact structure and lower model complexity.

A. Definition

The definition of the proposed SNPE is as follows.

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

where:

1) $O = \{a, e\}$ defines the alphabet over double objects where a stands for spikes (pulses) and e stands for enzymes.
2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ represent m neurons. $\sigma_i = (a_i, e_i, R_i)$, $1 \le i \le m$, where
   a. a_i and e_i are the original number of spikes and enzymes contained in σ_i, respectively.
   b. R_i means a finite set of rules. Rules involved include spiking rules and forgetting rules. The former is an extended spiking rule written as $E/(a^u, e^v) \rightarrow (a^p, e^q); d$, where E denotes a regular expression based on the alphabet $\{a\}$, d represents the time delay required to emit spikes when the rule fires. The forgetting rules do not require the participation of enzymes and have the form $a^s \rightarrow \lambda, s \ge 1$.
3) $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ represents the set of synapses. For $\forall (i, j) \in syn$, $1 \le i, j \le m$, $i \ne j$.
4) $in, out \in \{1, 2, \cdots, m\}$ correspond to the input and output neurons of the system, respectively.

The SNPE systems consist of three important components, namely, system objects, executing rules, and system structure. It can be seen from the above definition that an enzyme as a new object participates in all spiking rules. Since enzymes only exist inside nerve cells, they do not follow the information (spikes) to transmit between neurons. Furthermore, the enzymes involved in biochemical reactions (rules) can then be automatically synthesized in nerve cells for supplementation. In other words, in SNPE systems, a spiking rule can produce spikes and enzymes but only spikes are transmitted to external neurons while the enzymes still remain in the same neuron.

Suppose $u_i(t)$ is used to represent the number of spikes in σ_i at step t, and $v_i(t)$ is used to represent the number of enzymes in σ_i at step (time) t. At the next moment $(t + 1)$ after applying the rule $E/(a^u, e^v) \rightarrow (a^p, e^q); d$, the number of spikes and the number of enzymes in σ_i can be calculated by (2) and (3),

$$u_i(t+1) = \begin{cases} u_i(t) - u + n, & \text{If the rules fire} \\ u_i(t) + n, & \text{Other cases} \end{cases} \quad (2)$$

$$v_i(t+1) = \begin{cases} v_i(t) - v + q, & \text{If the rules fire} \\ v_i(t), & \text{Other cases} \end{cases} \quad (3)$$

The n in equation (2) represents the number of spikes that σ_i receives from its predecessor. u and v represent the number of spikes and enzymes needed to enforce the rule, respectively. p is the number of spikes generated by this rule, which are sent to the successor neurons along synaptic connections. q is the number of enzymes generated by this rule, which are only present in the current neuron to ensure the continued availability of the rule. d means the time delay between the use of the rule and the emission of spikes produced by that rule. For example, assuming rule $(a^u, e^v) \rightarrow (a^p, e^q); d$ in σ_i is enabled at step t, then the p spikes generated will leave σ_i at step $t + d$. From step t until step $t + d - 1$, σ_i is closed and is in the refractory period. That is, during these d steps,

σ_i can neither receive any spikes nor fire again. When d is not specified, its value is 0 by default.

The SNPE system can realize the parallel execution mode of a single rule in one neuron, which is similar to the exhaustive mode [18]. That is to say, if a neuron contains exactly an integer multiple (such as n) of the number of spikes and enzymes required by a rule, then this rule will be executed n times in parallel. For example, suppose there are 4 spikes (a) and 2 enzymes (e) in neuron σ_i, denoted as $[a^4, e^2]$. Then, rule $(a^2, e) \rightarrow (a, e)$ can be run twice in parallel at the same time. This rule will consume 4 spikes and 2 enzymes, then emit 2 produced spikes while also regenerating 2 enzymes.

In this work, we define the regular expression E of the rule as the trigger condition of this rule. For a rule to be enabled, it must ensure that both its trigger condition and its consumption condition are true, where the consumption condition means that there are enough spikes and enzymes in the neuron for the rule to consume. For example, if rule $E/(a^u, e^v) \rightarrow (a^p, e^q)$ in neuron σ_i can be enabled at step t, then the consumption condition of the rule must be true, meaning that the value of a Boolean expression of the form $(u_i(t) \geq u) \wedge (v_i(t) \geq v)$ must be 1. For convenience, we say that a rule satisfies its firing condition if it satisfies both its trigger condition and its consumption condition.

When multiple rules satisfy their firing conditions, the SNPE system follows the maximum spike-consumption strategy (mentioned in [30]). That is, when multiple rules are available, the one that consumes the largest number of spikes is selected for execution. For instance, if $u_1 > u_2$, then rule $E_1/(a^{u_1}, e^{v_1}) \rightarrow (a^{p_1}, e^{q_1})$ will be activated while rule $E_2/(a^{u_2}, e^{v_2}) \rightarrow (a^{p_2}, e^{q_2})$ will not be executed. In addition, the non-deterministic selection and execution of rules is also supported.

B. An Illustrative Example

This subsection introduces and explains the operating mechanism of the SNPE system through an illustrative example shown in Fig. 1. The demonstration case includes four neurons. The initial number of enzymes contained in each neuron is 2. Initially, only neuron 1 contains 2 spikes. Neuron 4 with an arrow pointing to the outside is marked as the output of the system. When the environment receives a spike sent by neuron 4, it can be regarded as the system output 1; otherwise, it outputs 0. When the computation of the system is terminated, the binary sequence (or spike train) sent by neuron 4 is the computational result. For the convenience of demonstration, the rules in this example temporarily omit the delay feature.

Let the rule in neuron 1 becomes active at time t. The number of spikes and enzymes contained in neuron 1 (written as σ_1) is exactly twice the number of spikes and enzymes required for the execution of the rule $(a^2)^+/(a, e) \rightarrow (a, e)$. Therefore, this rule in σ_1 will be executed twice in parallel simultaneously, and neuron 2 and 3 (denoted as σ_2, and σ_3, respectively) each receive 2 spikes. During this period, the enzymes consumed in the reaction are automatically synthesized in the neurons and restored to their original quantities.

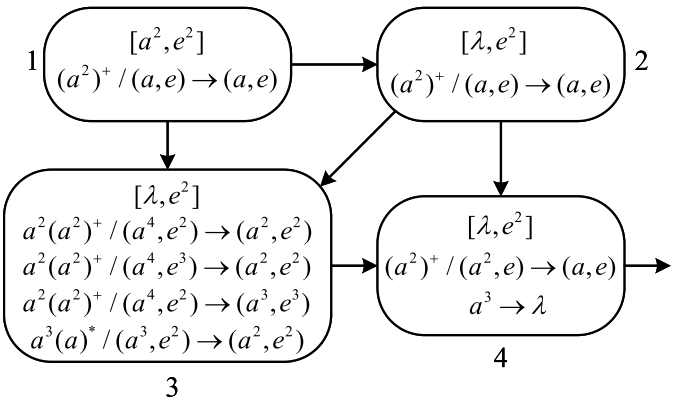

Fig. 1. An illustrative example of SNPE system.

After σ_2 receives two spikes, the situation is similar to σ_1 in the initial state. At $t + 1$, σ_2 fires 2 spikes to σ_3 and σ_4 respectively. After receiving the two spikes sent by σ_1, because the trigger condition is not met, no rule in σ_3 is activated. After neuron 4 (written as σ_4) receives the two spikes sent by σ_2, it satisfies the firing condition and sends out one spike for the first time at $t + 2$. Until another 2 spikes are obtained from σ_2, the four rules in σ_3 meet the trigger conditions. However, considering the maximum spike- consumption strategy, rule $a^3(a)^*/ (a^3, e^2) \rightarrow (a^2, e^2)$ cannot be enabled. Also, because σ_3 initially contains only two enzymes, rule $a^2(a^2)^+/ (a^4, e^3) \rightarrow (a^2, e^2)$ does not satisfy its consumption condition and therefore cannot be activated. The remaining rules $a^2(a^2)^+/ (a^4, e^2) \rightarrow (a^2, e^2)$ and $a^2(a^2)^+/ (a^4, e^2) \rightarrow (a^3, e^3)$ will be executed non-deterministically, which will cause the system to have different output results. Specifically, the following two scenarios will lead to completely different computations.

1) Assuming that rule $a^2(a^2)^+/(a^4, e^2) \rightarrow (a^2, e^2)$ is activated at $t+2$, σ_3 will fire 2 spikes to σ_4. σ_4 will run rule $(a^2)^+/(a^2, e) \rightarrow (a, e)$ at $t + 3$ and send out one spike outside again. At this time, the system terminates the computation, and the final result of the output binary sequence is "0011".
2) Assuming that rule $a^2(a^2)^+/(a^4, e^2) \rightarrow (a^3, e^3)$ is activated at $t + 2$, σ_3 will fire 3 spikes to σ_4. σ_4 will execute the forgetting rule $a^3 \rightarrow \lambda$ at $t + 3$. The 3 spikes received directly disappear and no spikes are launched outside. At this time, the system terminates the computation, and the final result of the output binary sequence is "0010".

IV. Turing Universality of SNPE Systems

As mentioned earlier, the simulation of the register machine is usually divided into two modes, that is, the generation mode and the acceptance mode. Each register r in M has a one-to-one correspondence with a neuron σ_r in SNPE systems. The number n stored in the register r corresponds to $2n$ spikes in the neuron σ_r. That is, there are twice as many spikes in the neuron σ_r as the value stored in the register r. Similarly, each instruction $l \in H$ corresponds to a neuron σ_l in SNPE systems one-to-one. In addition, there are some auxiliary neurons, denoted as σ_{b_i}. In the following, the Turing

universality of SNPE systems in these two modes is explored and proved. It should be noted that in the completeness proof of the following two modes, two spikes are introduced as the starting mechanism for all involved modules.

A. SNPE Systems as Number Generation Devices

In this work, the OUTPUT module in the generation mode takes the actual number of spikes emitted to the outside as the computational result. Under this assumption, the number of spikes emitted by the OUTPUT module to the outside will be exactly equal to the value stored in the register. Further, the family of $N_{gen}(\Pi_1)$ generated by the SNPE systems with the total number of neurons (m) involved and the maximum number of rules (n) contained in each neuron is denoted as $N_{gen}SNPE_m^n$.

Theorem 1: $N_{gen}SNPE_*^2 = \text{NRE}$

Proof: Since $N_{gen}SNPE_*^2 \subseteq NRE$ is obviously true (or, is achieved through Church-Turing thesis [4], [6]), to prove the above-mentioned equivalence relationship, we will only prove that $NRE \subseteq N_{gen}SNPE_*^2$ is also true below.

In the generating mode, an SNPE system Π_1 consists of 3 modules, ADD, SUB, and OUTPUT. When the neuron σ_{l_h} is activated, it means that the computation of the system is terminated. The OUTPUT module contains a neuron σ_{out} that can send spikes to the outside. Specifically, if the number stored in the register is n, then OUTPUT will cumulatively emit n spikes to the outside when the computation of the system Π_1 terminates.

ADD Module (see Fig. 2).

This module is used to simulate an addition instruction. The non-deterministic ADD module consists of 6 neurons, where σ_{l_i} is the trigger neuron of the addition instruction, σ_r is the register of the ADD instruction, σ_{b_1} and σ_{b_2} are auxiliary neurons, σ_{l_k} and σ_{l_j} represent jump instructions l_k and l_j respectively. Initially, only two spikes are introduced into the σ_{l_i}, and the remaining neurons do not contain any spikes. The amount of enzyme has been marked in individual neurons. Suppose that the neuron σ_{l_i} obtains 2 spikes at t, which denotes the addition instruction is activated. At this time, the number of spikes and enzymes contained in σ_{l_i} are two units respectively. This not only satisfies the firing condition of the rule $(a^2)^+/(a,e) \to (a,e);0$, but also makes the rule be executed twice in parallel simultaneously. The activated neuron σ_{l_i} simultaneously fires two spikes to σ_r, σ_{b_1}, and σ_{b_2}. The neuron σ_r after receiving two spikes marks that the register r has completed the addition operation.

At step $t+1$, both σ_{b_1} and σ_{b_2} contain 2 spikes, but only the rules in the neuron σ_{b_1} satisfy the firing condition. Since the two rules both satisfy the maximum spikes consumption strategy, only one rule in σ_{b_1} will be executed non- deterministically. As a result, the following two scenarios will appear.

a. Suppose that at $t+1$, rule $(a^2)^+/(a,e) \to (a,e);0$ is activated. σ_{b_1} will consume two spikes, and the generated two spikes will be transmitted to σ_{b_2} and σ_{l_k} simultaneously. The enzymes involved in the reaction are adjusted and changed according to equation (3). When σ_{l_k} gets 2 spikes, it marks the start of the execution of the instruction l_k. At $t+2$, 4 spikes will be accumulated in σ_{b_2}, but these 4 spikes will be immediately forgotten by the internal forgetting rule $a^4 \to \lambda$. In this scenario, σ_{l_j} will not receive any spikes, which means that the l_j instruction will not be activated and executed. Table I shows the computational process in the above scenario. The numbers in square brackets represent the number of spikes and enzymes at the corresponding step in the neuron, respectively.

b. Suppose that at $t+1$, the rule $(a^2)^+/(a^2, e) \to (a, e);$ 0 is activated. σ_{b_1} will fire a spike to σ_{b_2} and σ_{l_k} simultaneously. Similarly, the spike received by neuron σ_{l_k} will be directly consumed by its internal rule $a \to \lambda$. The instruction l_k cannot be executed. At $t+2$, the neuron σ_{b_2} has accumulated three spikes, which satisfies the firing condition of rule $a(a^2)^+/(a^3, e^2) \to (a^2, e^2);$ 0. At $t+3$, σ_{l_j} obtains 2 spikes, which marks the instruction l_j will be executed. The specific computational process under this scenario is presented in Table II.

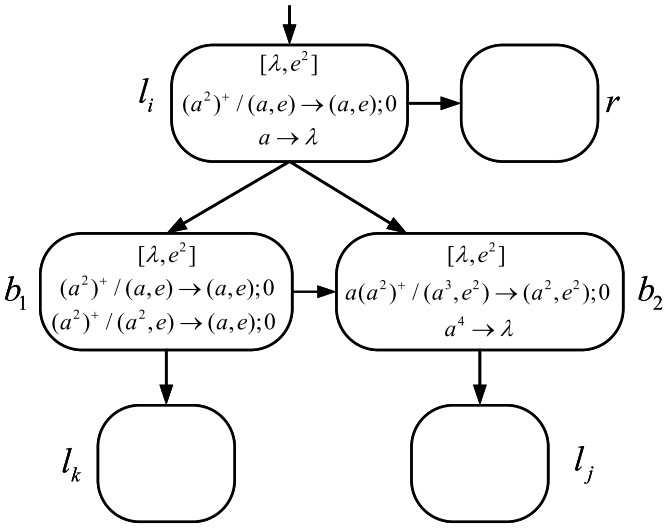

Fig. 2. Module ADD.

TABLE I
THE COMPUTATIONAL PROCESS UNDER SCENARIO A

Step	σ_{l_i}	σ_r	σ_{b_1}	σ_{b_2}	σ_{l_k}	σ_{l_j}
t	[2,2]	[2n,2]	[0,2]	[0,2]	[0,2]	[0,2]
t+1	[0,2]	[2n+2,2]	[2,2]	[2,2]	[0,2]	[0,2]
t+2	[2,2]	[2n+2,2]	[0,2]	[4,2]	[2,2]!	[0,2]

"!" indicates that the neuron is activated.

TABLE II
THE COMPUTATIONAL PROCESS UNDER SCENARIO B

Step	σ_{l_i}	σ_r	σ_{b_1}	σ_{b_2}	σ_{l_k}	σ_{l_j}
t	[2,2]	[2n,2]	[0,2]	[0,2]	[0,2]	[0,2]
t+1	[0,2]	[2n+2,2]	[2,2]	[2,2]	[0,2]	[0,2]
t+2	[0,2]	[2n+2,2]	[0,2]	[3,2]	[1,2]	[0,2]
t+3	[0,2]	[2n+2,2]	[0,2]	[0,2]	[0,2]	[2,2]!

"!" indicates that the neuron is activated.

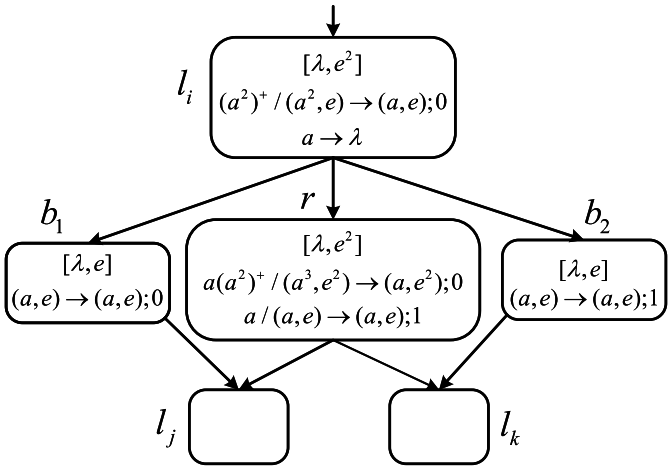

Fig. 3. Module SUB.

In summary, the addition instruction has been activated since σ_{l_i} received 2 spikes. Next, σ_r receives 2 spikes, which is equivalent to completing the operation of adding one to the existing value in r. Subsequently, by the non- deterministic execution of the two rules in the neuron σ_{b_1}, the simulation of the non-deterministic transfer of the two instructions l_k and l_j is realized.

SUB Module (see Fig. 3).

This module is developed to implement a subtraction instruction. The SUB module consists of 6 neurons, where σ_{l_i} is the trigger neuron of the subtraction instruction, σ_r corresponds to the register r of the SUB instruction, σ_{b_1} and σ_{b_2} are auxiliary neurons, σ_{l_k} and σ_{l_j} represent jump instructions l_k and l_j respectively. Initially, only two spikes are introduced into the σ_{l_i}, and the remaining neurons do not contain any spikes. The amount of enzyme has been marked in individual neurons. The initial scenario is similar to the previous addition instruction, i.e., it is assumed that 2 spikes enter σ_{l_i} at t, which means that the subtraction instruction starts to be simulated. At this time, rule $(a^2)^+/(a^2, e) \to (a, e); 0$ meets the firing condition, consumes two spikes, and a newly generated spike is transmitted to σ_{b_1}, σ_{b_2} and σ_r respectively. At $t+1$, σ_{b_1} executes rule $(a, e) \to (a, e); 0$ and fires 1 spike to σ_{l_j}. However, the next computational process of the system will be discussed in the following two cases according to the actual stored value in the register.

a. If the number existed in r is not empty (i.e., $n > 0$), that is, the number of spikes in σ_r is not less than 2. Then, at $t+1$, the spikes contained in σ_r is at least 3. At this time, only the rule $a(a^2)^+/(a^3, e^2) \to (a, e^2); 0$ will be activated and fire one spike to σ_{l_k} and σ_{l_j}, respectively. At $t+2$, σ_{l_j} will contain exactly 2 spikes, which means that instruction l_j will be activated and executed. In addition, because the rule in σ_{b_2} uses delay, σ_{l_k} will receive one spike from σ_r and σ_{b_2} at $t+2$ and $t+3$ respectively. That is, the two spikes that do not simultaneously arrive at σ_{l_k} will be forgotten separately. The specific evolution process of the SUB module in this scenario is presented in Table III.

TABLE III
THE COMPUTATIONAL PROCESS OF THE SUB UNDER SCENARIO A

Step	σ_{l_i}	σ_r	σ_{b_1}	σ_{b_2}	σ_{l_k}	σ_{l_j}
t	[2,2]	[2n,2]	[0,1]	[0,1]	[0,2]	[0,2]
t+1	[0,2]	[2n+1,2]	[1,1]	[1,1]*	[0,2]	[0,2]
t+2	[0,2]	[2n−2,2]	[0,1]	[1,1]*	[1,2]	[2,2]!

"*" indicates that the neuron uses the rule with delay in the corresponding step. "!" indicates that the neuron is activated.

TABLE IV
THE COMPUTATIONAL PROCESS OF THE SUB UNDER SCENARIO B

Step	σ_{l_i}	σ_r	σ_{b_1}	σ_{b_2}	σ_{l_k}	σ_{l_j}
t	[2,2]	[0,2]	[0,1]	[0,1]	[0,2]	[0,2]
t+1	[0,2]	[1,2]*	[1,1]	[1,1]*	[0,2]	[0,2]
t+2	[0,2]	[1,2]*	[0,1]	[1,1]*	[0,2]	[1,2]
t+3	[0,2]	[0,2]	[0,1]	[0,1]	[2,2]!	[0,2]

"*" indicates that the neuron uses the rule with delay in the corresponding step. "!" indicates that the neuron is activated.

b. If the number existed in the register r is empty (i.e., $n = 0$), that is, the number of spikes contained in σ_r is 0. At $t+1$, σ_r has only one spike sent from σ_{l_i}. At this time, only rule $a/(a, e) \to (a, e); 1$ will be activated and executed. Since the available rules in σ_r and σ_{b_2} both have the same time delay, at $t+3$, σ_{l_k} will receive one spike from σ_r and σ_{b_2}, respectively. This causes σ_{l_k} to be activated, meaning that the l_k instruction begins to be executed. Similar to the previous scenario, the single spike received by σ_{l_j} successively from σ_{b_1} and σ_r will be respectively forgotten. Table IV shows the computational process of the configuration in each neuron under this scenario.

In summary, since the neuron σ_{l_i} received 2 spikes, the subtraction instruction became active. When the number of spikes in σ_r is not empty, 2 spikes of them are consumed, and σ_{l_j} is further activated. Conversely, when the number of spikes in σ_r is zero, σ_{l_k} will be directly activated. The process successfully simulates the subtraction instruction.

OUTPUT Module (see Fig. 4).

This module is designed to simulate the termination instruction l_h, and yields the system's computation results. Suppose σ_{l_h} gets 2 spikes at step t, marking the start of the termination instruction to be simulated. Rule $(a^2, e) \to (a, e); 0$ is executed, and 1 spike is fired to σ_1.

Assuming that the value existed in register 1 is n, correspondingly, the number of spikes in σ_1 is $2n$. Then, after neuron σ_1 obtains one spike at $t+1$, the number of spikes it contains will become odd. Rule $a^3(a^2)^+/(a^2, e) \to (a, e); 0$ in σ_1 is activated and executed. It should be noted that the regular expression for this rule requires that the number of spikes contained in σ_1 cannot be less than 5. This is to avoid a conflict with the applicability of another rule.

From $t+1$ to $t+n-1$, $2(n-1)$ spikes will be consumed cumulatively. At $t+n$, only 3 spikes remain in σ_1. At this point, rule $(a^3, e^2) \to (a, e^2); 0$ is enabled, consuming 3 spikes and emitting one spike to σ_{out}. On the other hand,

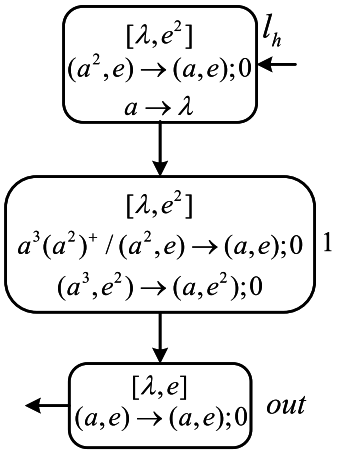

Fig. 4. Module OUTPUT.

TABLE V
EVOLUTION PROCESS OF OUTPUT MODULE

Step	σ_{l_h}	σ_1	σ_{out}
t	[2,2]	$[2n,2]$	[0,1]
$t+1$	[0,2]	$[2n+1,2]$	[0,1]
$t+2$	[0,2]	$[2n-1,2]$	[1,1]!
$t+3$	[0,2]	$[2n-3,2]$	[1,1]!
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t+n$	[0,2]	[3,2]	[1,1]!
$t+n+1$	[0,2]	[0,2]	[1,1]!

"!" indicates that the neuron is activated.

σ_{out} accumulatively receives n spikes transmitted by σ_1 from $t+2$ to $t+n+1$, and transmits them to the outside in turn. Table V clearly shows the above evolution process. The first column represents the time step. The other three columns are the number of spikes and enzymes contained in the three neurons.

Through the above analysis, it can be easily found that the generation mode can be successfully simulated by Π_1. Therefore, Theorem 1 is proved to be true. □

B. SNPE Systems as Number Acceptance Devices

Theorem 2: $N_{acc}SNPE_*^2 = \text{NRE}$

Proof: In this proof we build an SNPE system Π_2 to realize the acceptance mode. The proof process of the accepting mode is somewhat similar to that of the generation mode. The difference is that the system Π_2 in this mode is composed of INPUT, deterministic ADD, and SUB. Among them, the SUB module still uses the structure of Fig. 3. It should be noted that, considering all the rules in the INPUT module and the ADD module that need to be proved in this subsection do not involve delay, all the rules ignore delay for the sake of convenience. The specific reasoning is proved as follows.

INPUT Module (see Fig. 5).

The purpose of this module is to import binary sequence information from the outside. Initially, only two spikes are introduced into the σ_{in}, and the remaining neurons do not contain any spikes. Suppose σ_{in} gets 2 spikes at t, and the firing condition of rule $(a^2)^+/(a, e) \to (a, e)$ is satisfied and

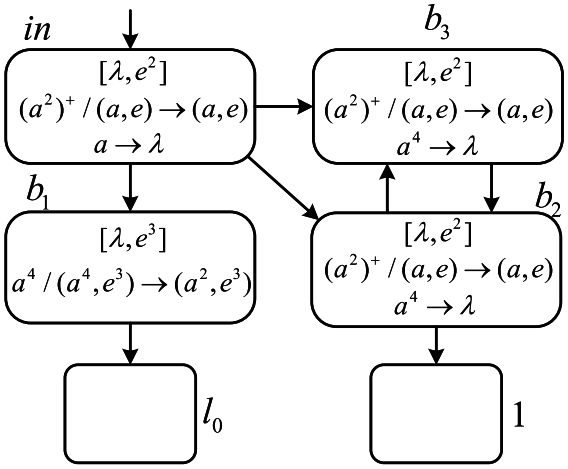

Fig. 5. Module INPUT.

activated. This rule is executed twice in parallel simultaneously, and 2 spikes are fired to σ_{b_1}, σ_{b_2}, and σ_{b_3}, respectively. At $t+1$, although σ_{b_1} obtains 2 spikes, it does not meet the firing conditions of the rules contained in it. Therefore, its internal rules cannot be enabled. Simultaneously, after the neurons σ_{b_2} and σ_{b_3} receive two spikes, the rule $(a^2)^+/(a, e) \to (a, e)$ within them is activated, and from this moment on, they send two spikes to each other at the same time. At $t+2$, σ_1 gets and stores 2 spikes for the first time. At $t+n$, σ_{in} obtains 2 pulses from the outside again. At $t+n+1$, the neurons σ_{b_2} and σ_{b_3} will hold four spikes at the same time, which meets the forgetting rule $a^4 \to \lambda$. The four spikes accumulated in neurons σ_{b_2} and σ_{b_3} are all forgotten at this moment. Therefore, at $t+n+1$, it is the last time that neuron σ_1 gets two spikes from σ_{b_2}. From $t+2$ to $t+n+1$, σ_1 has accumulated $2n$ pulses. This simulates the value existed in register 1 is n, and n also happens to be the time difference between the two pulses. On the other side, at $t+n+1$, σ_{b_1} obtained two spikes from σ_{in} for the second time. Therefore, the number of spikes accumulated inside is four, rule $a^4/(a^4, e^3) \to (a^2, e^3)$ is activated, and 2 spikes are fired to σ_{l_0}. After σ_{l_0} gets 2 spikes, it indicates that the initial instruction l_0 will be simulated.

ADD Module (see Fig. 6).

This module realizes the simulation of deterministic addition instructions. Suppose that the neuron σ_{l_i} obtains 2 spikes at step t. At this time, rule $(a^2)^+/(a, e) \to (a, e)$ fires 2 spikes to σ_r and σ_{l_j} respectively. After σ_r gets 2 spikes, it indicates the existing storage value in the register r is increased by 1. Simultaneously, σ_{l_j} received 2 spikes, which corresponds to the deterministic execution of the instruction l_j.

Through the above deduction, it can be clearly found that the computation in this mode can be successfully simulated by Π_2. Hence, Theorem 2 is proved to be true. □

V. SNPE SYSTEMS AS FUNCTION COMPUTING DEVICES

Based on the analysis and discussion in the previous section, this section will explore the computing power of the SNPE system as a function computing device. First consider a Turing

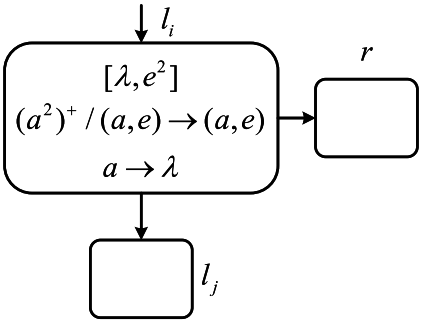

Fig. 6. Module ADD.

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

Fig. 7. A universal register machine M_u.

computable function $f : N^k \to N$, in which k parameters are stored in designated k registers. The computation of the system starts with l_0 and ends with l_h. For $\forall x, y \in N$, if there is a recursive function g that satisfies the equation $\varphi_x(y) = M_u(g(x), y)$, it is said that M_u is universal.

Korec [47] once proposed a universal register machine M_u for function computing (see Fig. 7). In order to avoid the possibility of the final computation result stored in register 0 being affected by the SUB instruction, the following minor improvements are made to M_u. A new register 8 which is not affected by the SUB instruction is added to store the final computation result. The original instruction l_{18} is changed to $l_{18} : (SUB(4), l_0, l_{22})$. Expand the last l_h to $l_h : HALT$, $l_{22} : (SUB(0), l_{23}, l_h)$, and $l_{23} : (ADD(8), l_{22})$.

Theorem 3: There exists a general SNPE system with 61 neurons that implements a computable function.

Proof: Mark the M_u after fine adjustment as M'_u. This part will build an SNPE system Π_3 composed of 1 INPUT module, 1 OUTPUT module, 10 ADD modules, and 14 SUB modules to realize the simulation of M'_u. As mentioned before, the INPUT module (see Fig. 8) is still used to read binary sequence information from the environment. The binary sequence that the system needs to recognize from the outside is $10^{g(x)}10^y1$. The expected situation is that the values $g(x)$ and y are stored in registers 1 and 2, respectively, and then the initial instruction l_0 is started. It should be noted that, in the previous completeness proof of generation devices

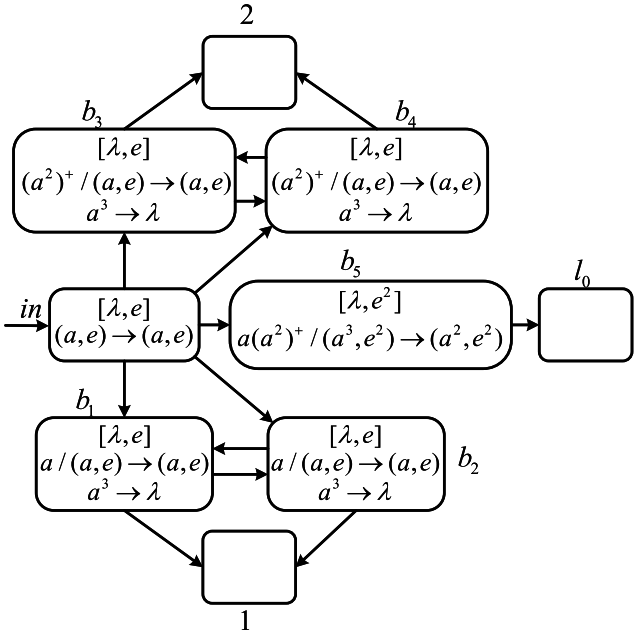

Fig. 8. Module INPUT of SNPE systems.

and acceptance devices, for the sake of formal unification, we introduce two spikes as the starting mechanism for the proof of all involved modules. However, the function computing devices considered in this section need to redesign the INPUT module according to its characteristics. Therefore, slightly different from the input mechanism of the INPUT module in acceptance mode, the INPUT module here uses a single spike to correspond to 1 in the binary sequence. In addition, the delay mechanism is not involved in this module, so it is ignored.

The working mechanism of this module is stated as follows. Initially, only σ_{in} gets the first one pulse from the outside. Its internal rule $(a, e) \to (a, e)$ is triggered, and fires 1 spike to each of the five subsequent auxiliary neurons simultaneously. But only the rule $a/(a, e) \to (a, e)$ in σ_{b_1} and σ_{b_2} meets the firing condition and is activated. At each step from this time on, σ_{b_1} and σ_{b_2} will complement each other with one spike, and both send a spike to σ_1 simultaneously. Until σ_{b_1} and σ_{b_2} receive the second spike from σ_{in}, during this period, σ_1 will accumulatively obtain $2g(x)$ spikes.

When σ_{in} gets the second spike from the outside, $(a, e) \to (a, e)$ is triggered again and simultaneously fires 1 spike to each of the five auxiliary neurons. At this time, none of the rules in the neurons σ_{b_1}, σ_{b_2}, and σ_{b_5} satisfy the firing condition, so none of them can be activated. However, two spikes have been accumulated in neurons σ_{b_3} and σ_{b_4}, satisfying the firing condition of $(a^2)^+/(a, e) \to (a, e)$. At each step from this moment on, the neurons σ_{b_3} and σ_{b_4} will consume and complement each other with one spike, and both send one spike to σ_2 simultaneously. It should be noted that during this process, 2 spikes will always remain in σ_{b_3} and σ_{b_4} due to the complementary mechanism. Until σ_{b_3} and σ_{b_4} receive the third spike from σ_{in}, during this period, neuron σ_2 will accumulate $2y$ spikes.

$$
\begin{aligned}
&\text{SUB-ADD-1}\left\{\begin{array}{ll}
l_0:(SUB(1),l_1,l_2); & l_1:(ADD(7),l_0)\\
l_4:(SUB(6),l_5,l_3); & l_5:(ADD(5),l_6)\\
l_6:(SUB(7),l_7,l_8); & l_7:(ADD(1),l_4)\\
l_8:(SUB(6),l_9,l_0); & l_9:(ADD(6),l_{10})\\
l_{14}:(SUB(5),l_{16},l_{17}); & l_{16}:(ADD(4),l_{11})\\
l_{22}:(SUB(0),l_{23},l_h); & l_{23}:(ADD(8),l_{22})
\end{array}\right.\\
&\text{SUB-ADD-2}\left\{\begin{array}{ll}
l_{15}:(SUB(3),l_{18},l_{20}); & l_{20}:(ADD(0),l_0)
\end{array}\right.\\
&\text{ADD-ADD}\left\{\begin{array}{ll}
l_{17}:(ADD(2),l_{21}); & l_{21}:(ADD(3),l_{18})
\end{array}\right.
\end{aligned}
$$

Fig. 9. Compound instruction optimization scheme.

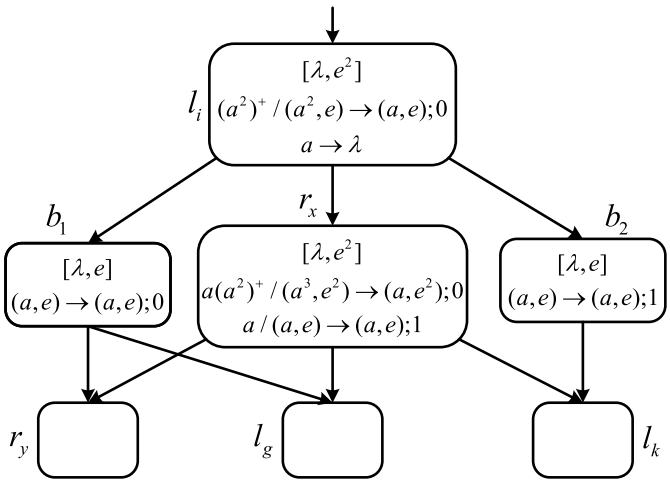

Fig. 10. Module $SUB-ADD_1$.

When σ_{in} gets the third spike from the outside and is activated, the neurons σ_{b_1}, σ_{b_2}, σ_{b_3}, and σ_{b_4} will accumulate to three spikes. However, 3 spikes of the above four neurons will be immediately forgotten by their internal forgetting rule $a^3\to\lambda$. At this time, only the rule $a(a^2)^+/(a^3,e^2)\to(a^2,e^2)$ in σ_{b_5} is finally activated, consuming 3 spikes and sending 2 spikes to σ_{l_0}. This marks the successful end of obtaining data from the environment and the start of the system's initial instruction l_0 to be executed.

It can be seen from Fig. 7 that all ADD modules that need to be simulated are deterministic. Therefore, the ADD module adopts the form of Fig. 6. Refer to Fig. 3 for the SUB module. Refer to Fig. 4 for the OUTPUT module. The statistics of neurons required by various instructions or modules in the SNPE system Π_3 to simulate the function computing are as follows.

a) 9 registers correspond to 9 neurons;
b) 25 instructions are associated with 25 neurons;
c) 14 SUB instructions correspond to 2×14 neurons;
d) An INPUT module contains 6 neurons;
e) One OUTPUT module corresponds to one neuron;

In summary, in the above simulation process, a total of 69 neurons were used. However, after considering the possible merging of some modules and the necessary optimization design of their structure, the number of neurons used in the system will be much reduced. According to the compound instruction optimization scheme (see Fig. 9), the combination of instruction *SUB* and instruction *ADD* can be divided into three situations: *SUB-ADD*$_1$, *SUB-ADD*$_2$ and *ADD-ADD*.

Among them, the general expression form of $SUB-ADD_1$ (see Fig. 10) type can be written as $l_i:(SUB(r_x),l_j,l_k)$, $l_j:(ADD(r_y),l_g)$. The first six combinations shown in Fig. 9 conforming to the $SUB-ADD_1$ type have a common law, that is, when the value existed in the register r_1 of the previous instruction is not empty, it will jump directly and execute the latter instruction. The situation with $SUB-ADD_2$ (see Fig. 11) is just the opposite. It should be emphasized that the initial configuration of the merge modules $SUB-ADD_1$ and $SUB-ADD_2$ is exactly the same as that of the *SUB* module.

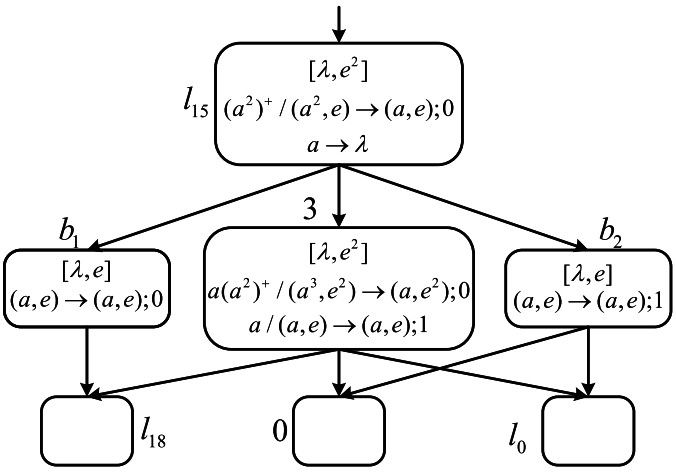

Fig. 11. Module $SUB-ADD_2$.

We take Fig. 10 as an example to give the evolution process of the $SUB-ADD_1$ module. It should be emphasized that this type of module has a common feature, that is, the first output instruction of SUB is exactly the input instruction of ADD. The initial scenario of this module is similar to the SUB module shown in Fig. 3. The difference is that the rules with 0 delay in σ_{r_x} and σ_{b_1} will send one spike to σ_{r_y} and σ_{l_g} simultaneously when register r_x of instruction l_i is not empty. In other words, σ_{r_y} and σ_{l_g} will receive two spikes simultaneously. For instruction l_j, this is equivalent to simultaneously completing the two operations of adding 1 to the stored value of register r_y and deterministically going to instruction l_g. The single spike sent from σ_{r_x} and σ_{b_2} to σ_{l_k} will be forgotten successively in σ_{l_k} due to the time interval. In addition, the rules with 1 delay in σ_{r_x} and σ_{b_2} will be fired simultaneously when the value stored in register r_x is empty. In this scenario, σ_{l_k} will receive 2 spikes simultaneously, which means that the instruction l_k starts to execute. Each of the 6 combinations that can be classified as $SUB-ADD_1$ saves one neuron. The evolution process of the *SUB-ADD*$_1$ module when the register r_x is not empty is shown in Table VI. Table VII shows

TABLE VI
THE EVOLUTION PROCESS OF THE *SUB-ADD*$_1$ MODULE WHEN THE REGISTER r_x IS NOT EMPTY

Step	σ_{l_i}	σ_{r_x}	σ_{b_1}	σ_{b_2}	σ_{r_y}	σ_{l_g}	σ_{l_k}
t	[2,2]	[2n,2]	[0,1]	[0,1]	[0,2]	[0,2]	[0,2]
t+1	[0,2]	[2n+1,2]	[1,1]	[1,1]*	[0,2]	[0,2]	[0,2]
t+2	[0,2]	[2n−2,2]	[0,1]	[1,1]*	[2,2]!	[2,2]!	[1,2]

"*" indicates that the neuron uses the rule with delay in the corresponding step. "!" indicates that the neuron is activated.

TABLE VII
THE EVOLUTION PROCESS OF THE *SUB-ADD*$_2$ MODULE WHEN THE REGISTER 3 IS EMPTY

Step	$\sigma_{l_{15}}$	σ_3	σ_{b_1}	σ_{b_2}	σ_0	$\sigma_{l_{18}}$	σ_{l_0}
t	[2,2]	[0,2]	[0,1]	[0,1]	[0,2]	[0,2]	[0,2]
t+1	[0,2]	[1,2]*	[1,1]	[1,1]*	[0,2]	[0,2]	[0,2]
t+2	[0,2]	[1,2]*	[0,1]	[1,1]*	[0,2]	[1,2]	[0,2]
t+3	[0,2]	[0,2]	[0,1]	[0,1]	[2,2]!	[1,2]	[2,2]!

"*" indicates that the neuron uses the rule with delay in the corresponding step. "!" indicates that the neuron is activated.

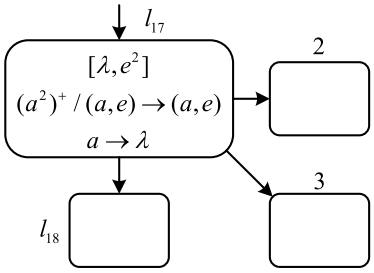

Fig. 12. Module *ADD* − *ADD*.

TABLE VIII
COMPARISON OF DIFFERENT COMPUTING MODELS

Model name	#neurons	Maximum #rules
SNPE	61	2
DTNP[23]	109	3
NSNP[32]	117	3
DeP[48]	115	2
SNP-IR[33]	100	3
IR-SSNP[49]	89	3
DSNP[50]	81	4
SNPA[51]	75	3

the computational process of the *SUB-ADD*$_2$ module when register 3 is empty.

In addition, because the ADD instruction is deterministic, the instructions l_{17} and l_{21} can be combined in the form of an $ADD-ADD$ (see Fig. 12) module. According to this merging optimization, the neuron corresponding to the instruction l_{21} can be saved.

The above three types contain a total of 8 combinations, and each combination can save one neuron. Therefore, 8 neurons will be removed due to redundancy. Table VIII shows the total number of neurons required and the maximum number of rules

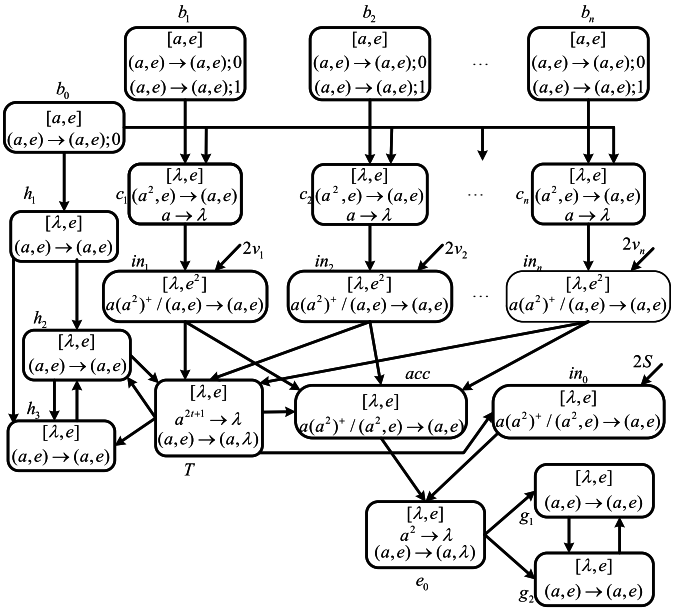

Fig. 13. Construction of the SNPE model for solving the subset sum problem.

presented in each neuron when different computing models are used as universal function computing devices. It can be seen that our model (SNPE) can use a smaller number of neurons and only two rules are involved in each neuron. In summary, after combining and optimizing related modules, the total number of neurons required can be reduced from 69 to 61. Therefore, the statement of the Theorem 3 is obviously true. □

VI. UNIFORM SOLUTION TO SUBSET SUM PROBLEM

A. The Subset Sum Problem

The subset sum problem is NP-complete and can be described as follows. Given a set V of n positive integers and a positive integer S, is there a subset $B \subseteq V$ such that the sum of all elements in B is exactly equal to S? Leporati *et al.* [52] and Leporati *et al.* [53] have used standard SN P systems to solve subset sum problem in a non-uniform and uniform manner, respectively. Non-uniform way relies on specific instances to design models, whereas a uniform way is problem-oriented rather than concrete instances. In other words, solving the subset sum problem in a uniform manner means that the design of the system depends only on the size n of the problem, while the specific elements of V and S need to be introduced into the system. Obviously, the uniform way is more transparent than the non-uniform way [53]. Therefore, this work explores the use of SNPE systems to solve subset sum problem in a uniform way.

B. Model Construction

The construction of the SNPE model to solve the subset sum problem is shown in Fig. 13. We follow the design philosophy of Leporati *et al.* [53] that if the problem is solved then the computation of the system stops automatically, otherwise the

computation of the system continues forever. The specific reasoning process is as follows.

There are $n+1$ input neurons in the model, denoted as in_0, in_1,..., in_n, which are responsible for introducing $2S$, $2v_1$, $2v_2$, ..., $2v_n$ (where, $V=\{v_1, v_2, \ldots, v_n\}$) spikes into the system, respectively. Initially, only one spike exists in neurons σ_{b_0}, $\sigma_{b_1}, \ldots, \sigma_{b_n}$, and the rest of the neurons do not contain any spikes. It should be noted that, according to the design requirements of the problem, we only involve different delay rules in σ_{b_0}, $\sigma_{b_1}, \ldots, \sigma_{b_n}$. It is through the non-deterministic choice of these rules with different time delays that the non-deterministic choice of positive integers in V is achieved. Suppose that at step 1, neurons σ_{b_1},..., σ_{b_n} begin to non-deterministically select one rule within them to execute. For example, assuming that σ_{b_i} non-deterministically executes rule $(a, e) \rightarrow (a, e); 0$ at step 1, then at step 2, σ_{c_i} will receive a single spike from σ_{b_0} and σ_{b_i} simultaneously. This causes rule $(a^2, e) \rightarrow (a, e)$ in σ_{c_i} to be activated. Next, the input neuron σ_{in_i} will receive one spike at step 3, which makes the number of spikes in it become odd (i.e., $2v_i+1$). Note that the intrinsic number of enzymes in σ_{in_i} is 2, which allows its internal rule $a(a^2)/(a, e) \rightarrow (a, e)$ to be executed at most twice simultaneously and sends 2 spikes to the accumulating neuron σ_{acc} and trigger neuron σ_T. However, the number of spikes remaining in σ_{in_i} is still odd, which causes its internal rules to be executed continuously for v_i steps. In other words, from step 4 to step $4+(v_i-1)$, the neuron σ_{acc} will continuously receive a total of $2v_i$ spikes from σ_{in_i}. In this way, a positive integer v_i in the set V is non-deterministically selected and imported into σ_{acc} in the form of $2v_i$ spikes.

On the other hand, at step 4, the trigger neuron σ_T receives not only an even number of spikes from σ_{in_i}, but also a single spike from σ_{h_2}. It should be emphasized that the larger the value of the non-deterministically chosen number v_i, the more steps are required to transfer $2v_i$ spikes from σ_{in_i} to σ_{acc}. Given this concern, the trigger neuron σ_T is therefore designed to determine whether all selected numbers have been sent to the σ_{acc}. The rule $a^{2t+1} \rightarrow \lambda$ in σ_T is available as long as the transfer of spikes from σ_{in_i} to σ_{acc} is not over. When σ_T no longer receives spikes from any $\sigma_{in_i} (1 \leq i \leq n)$, it means that all non-deterministically chosen numbers have been stored in σ_{acc}. At this point, σ_T will only receive one spike from σ_{h_2}, causing rule $(a, e) \rightarrow (a, \lambda)$ to be enabled. It is important to note that this rule consumes one enzyme and no new enzymes are produced to replenish. This means that rule $(a, e) \rightarrow (a, \lambda)$ will no longer be available from now on. That is, when the enzyme is not renewable, the execution of this rule is no longer sustainable. After σ_{h_2} and σ_{h_3} receive a spike from σ_T, the number of spikes in them is 2, causing the rules in them to stop executing.

When σ_{acc} and σ_{in_0} receive one spike from σ_T, the number of spikes contained will become odd. Then the only rule within σ_{acc} and σ_{in_0} will be activated, sending one spike to σ_{e_0} while consuming two of its own spikes. After σ_{e_0} receives two spikes, it immediately forgets them according to rule $a^2 \rightarrow \lambda$. The above process continues until one of the following three scenarios occurs. (i) The most optimistic scenario must be that σ_{acc} and σ_{in_0} contain exactly the same number of spikes. In this case, after σ_{e_0} executes the forgetting rule for the last time, the entire system will stop computing. This means that the sum of all numbers of non-deterministic choices is exactly equal to S, and the subset sum problem is solved. (ii) The number of spikes stored in σ_{acc} is more than that stored in σ_{in_0}. (iii) Conversely, more spikes are stored in σ_{in_0} than in σ_{acc}. The latter two scenarios can be grouped into one category, that is, σ_{e_0} will start to execute rule $(a, e) \rightarrow (a, \lambda)$ after executing rule $a^2 \rightarrow \lambda$ for the last time. As mentioned earlier, rule $(a, e) \rightarrow (a, \lambda)$ consumes one enzyme without reproducing it, which causes this rule will no longer be available from now on. Subsequently, σ_{g_1} and σ_{g_2} will work forever.

As can be seen from Fig. 13, the uniform SNPE model requires only $3n+10$ neurons. And at most $2\sum_{i=1}^{n} v_i + 3$ steps are required when the computation of this model is stopped automatically, including the initial 2 steps, from σ_{in_i} to σ_{acc} at most $\max_{1 \leq i \leq n} \{v_i\}$ steps, from σ_{acc} (or σ_{in_0}) to σ_{e_0} at most $\sum_{i=1}^{n} v_i$ steps, and one step from σ_T to σ_{acc} (and σ_{in_0}). Considering that $\max_{1 \leq i \leq n} \{v_i\} \leq \sum_{i=1}^{n} v_i$ must be true, we take $\sum_{i=1}^{n} v_i$ as the upper bound of $\max_{1 \leq i \leq n} \{v_i\}$. Therefore, when the calculation of the system is automatically terminated, at most $2\sum_{i=1}^{n} v_i + 3$ steps are required.

Finally, we compare with the standard SN P [53] and its several state-of-the-art variants in terms of complexity (see Table IX). It is evident from Table IX that the original SN P has the largest number of neurons used and the number of steps required to compute termination. Our SNPE is second only to RSSN P [54] in terms of using the total number of neurons, but outperforms the other three compared models. Whereas, the proposed SNPE model outperforms all four compared models in terms of the number of steps required.

TABLE IX
MODEL COMPLEXITY COMPARISON

Model	neurons	steps
SN P[53]	$5n+13$	$3\sum_{i=1}^{n} v_i + 6$
time-free SN P[55]	$5n+2$	$3\sum_{i=1}^{n} v_i + 2$
SNPSP[56]	$4n+9$	$2\sum_{i=1}^{n} v_i + 6$
RSSN P[54]	$2n+11$	$2\sum_{i=1}^{n} v_i + 5$
SNPE	$3n+10$	$2\sum_{i=1}^{n} v_i + 3$

VII. CONCLUSION

This work proposes new spiking neural P systems with enzymes (SNPE). Compared with the standard SN P and its variants, SNPE has made improvements in terms of objects, rules, and system operation. While giving the formal definition of the SNPE systems, it also gives the expression of the change in the number of two kinds of objects in the neuron. Turing computing ability of the proposed SNPE systems in generation mode and acceptance mode is proved respectively. The computing power of this system as a small universal function computing device is demonstrated and compared

with 7 SN P variants. Finally, the performance of the system in solving NP-complete problem is explored and compared with the standard SN P and its several state-of-the-art variants.

It should be emphasized that the excellent performance (see Table VIII) of the proposed SNPE as a small universal function computing device is not independently and directly contributed by the "enzyme", but the collaboration of multiple mechanisms including the "delay". There are indeed some SN P variants that use a smaller total number of neurons than SNPE when simulating a function computing device. For example, SN P with request rules [57] uses only 47 neurons, yet the number of rules in its neurons is as high as 11, which is much higher than all the comparison models in Table VIII. In addition, it is precisely because of the reasonable control of the "enzyme" over the sustainability of rule execution that SNPE excels in solving the Subset Sum problem.

Based on this work, a lot of follow-up work can be explored and carried out. On the one hand, it can be combined with existing SN P variants. These combinations may lead to the emergence of other systems with more computing power. On the other hand, considering the control and restriction effect of enzymes on nerve conduction, other variants of SNPE systems can be developed. Finally, we can jump out of the biological principle and consider making some new expansions toward the existing neural network in terms of the topology of SNPE.

REFERENCES

[1] G. Paŭn, "Computing with membranes," *J. Comput. Syst. Sci.*, vol. 61, no. 1, pp. 108–143, Aug. 2000.

[2] G. Păun, G. Rozenberg, A. Salomaa, and O. U. Press, Ed., *The Oxford Handbook of Membrane Computing*. Oxford, U.K.: Oxford Univ. Press, 2010.

[3] Z. Lv *et al.*, "Spiking neural P systems with extended channel rules," *Int. J. Neural Syst.*, vol. 31, no. 1, pp. 1–13, Jan. 2021.

[4] M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural P systems," *Fundam. Inf.*, vol. 71, no. 2, pp. 279–308, 2006.

[5] G. Păun, "Spiking neural P systems with astrocyte-like control," *J. Univ. Comput. Sci.*, vol. 13, no. 11, pp. 1707–1721, 2007.

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

[7] B. Aman and G. Ciobanu, "Spiking neural P systems with astrocytes producing calcium," *Int. J. Neural Syst.*, vol. 30, no. 12, pp. 1–16, Dec. 2020.

[8] L. Pan and G. Păun, "Spiking neural P systems with anti-spikes," *Int. J. Comput., Commun., Control*, vol. 4, no. 3, pp. 273–282, Sep. 2009.

[9] T. Song, X. Liu, Y. Zhao, and X. Zhang, "Spiking neural P systems with white hole neurons," *IEEE Trans. Nanobiosci.*, vol. 15, no. 7, pp. 666–673, Oct. 2016.

[10] T. Wu, A. Păun, Z. Zhang, and L. Pan, "Spiking neural P systems with polarizations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3349–3360, Aug. 2018.

[11] T. Wu and L. Pan, "The computation power of spiking neural P systems with polarizations adopting sequential mode induced by minimum spike number," *Neurocomputing*, vol. 401, pp. 392–404, Aug. 2020.

[12] H. Peng *et al.*, "Spiking neural P systems with multiple channels," *Neural Netw.*, vol. 95, pp. 66–71, Nov. 2017.

[13] Y. Huang *et al.*, "On string languages generated by spiking neural P systems with multiple channels," *Int. J. Unconventional Comput.*, vol. 14, nos. 3–4, pp. 243–266, 2019.

[14] M. Cavaliere, O. H. Ibarra, G. Păun, O. Egecioglu, M. Ionescu, and S. Woodworth, "Asynchronous spiking neural P systems," *Theor. Comput. Sci.*, vol. 410, no. 24, pp. 2352–2364, May 2009.

[15] L. Pan, J. Wang, and H. J. Hoogeboom, "Limited asynchronous spiking neural P systems," *Fundamenta Informaticae*, vol. 110, nos. 1–4, pp. 271–293, 2011.

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

[17] M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural P systems with an exhaustive use of rules," *Int. J. Unconv. Comput.*, vol. 3, no. 2, pp. 135–153, 2007.

[18] L. Pan and X. Zeng, "Small universal spiking neural P systems working in exhaustive mode," *IEEE Trans. Nanobiosci.*, vol. 10, no. 2, pp. 99–105, Jun. 2011.

[19] L. Pan, G. Paun, and B. Song, "Flat maximal parallelism in P systems with promoters," *Theor. Comput. Sci.*, vol. 623, pp. 83–91, Apr. 2016.

[20] B. Song, M. J. Pérez-Jiménez, G. Păun, and L. Pan, "Tissue P systems with channel states working in the flat maximally parallel way," *IEEE Trans. Nanobiosci.*, vol. 15, no. 7, pp. 645–656, Oct. 2016.

[21] T. Wu and S. Jiang, "Spiking neural P systems with a flat maximally parallel use of rules," *J. Membrane Comput.*, vol. 3, no. 3, pp. 221–231, Sep. 2021.

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

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

[24] Y. Huang, W. Yi, H. Peng, J. Wang, X. Luo, and Q. Yang, "Computational power of dynamic threshold neural P systems for generating string languages," *Theor. Comput. Sci.*, vol. 851, pp. 77–91, Jan. 2021.

[25] J. Wang, H. J. Hoogeboom, L. Pan, G. Păun, and M. J. Pérenz-Jiménez, "Spiking neural P systems with weights," *Neural Comput.*, vol. 22, no. 10, pp. 2615–2646, 2010.

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

[27] F. G. C. Cabarle, H. Adorna, M. J. Pérez-Jiménez, and T. Song, "Spiking neural P systems with structural plasticity," *Neural Comput. Appl.*, vol. 26, no. 8, pp. 1905–1917, 2013.

[28] 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," *J. Membrane Comput.*, vol. 3, no. 1, pp. 10–21, Feb. 2021.

[29] X. Wang, T. Song, F. Gong, and P. Zheng, "On the computational power of spiking neural P systems with self-organization," *Sci. Rep.*, vol. 6, no. 1, pp. 1–16, Jun. 2016.

[30] H. Peng and J. Wang, "Coupled neural P systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1672–1682, Jun. 2019.

[31] T. Wu, L. Pan, Q. Yu, and K. C. Tan, "Numerical spiking neural P systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 6, pp. 1–15, Jul. 2020.

[32] H. Peng *et al.*, "Nonlinear spiking neural P systems," *Int. J. Neural Syst.*, vol. 30, no. 10, pp. 1–17, 2020.

[33] H. Peng *et al.*, "Spiking neural P systems with inhibitory rules," *Knowl.-Based Syst.*, vol. 188, Jan. 2020, Art. no. 105064.

[34] T. Neary, "Three small universal spiking neural P systems," *Theor. Comput. Sci.*, vol. 567, pp. 2–20, Feb. 2015.

[35] X. Zhang, X. Zeng, and L. Pan, "Smaller universal spiking neural P systems," *Fundamenta Informaticae*, vol. 87, no. 1, pp. 117–136, Feb. 2008.

[36] Q. Ren, X. Liu, and M. Sun, "Turing universality of weighted spiking neural P systems with anti-spikes," *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–10, Sep. 2020.

[37] T. Wu, L. Pan, and A. Alhazov, "Computation power of asynchronous spiking neural P systems with polarizations," *Theor. Comput. Sci.*, vol. 777, pp. 474–489, Jul. 2019.

[38] X. Song, H. Peng, J. Wang, G. Ning, and Z. Sun, "Small universal asynchronous spiking neural P systems with multiple channels," *Neurocomputing*, vol. 378, pp. 1–8, Feb. 2020.

[39] Q. Yang, B. Li, Y. Huang, H. Peng, and J. Wang, "Spiking neural P systems with structural plasticity and anti-spikes," *Theor. Comput. Sci.*, vol. 801, pp. 143–156, Jan. 2020.

[40] L. Pan, T. Wu, Y. Su, and A. V. Vasilakos, "Cell-like spiking neural P systems with request rules," *IEEE Trans. Nanobiosci.*, vol. 16, no. 6, pp. 513–522, Sep. 2017.

[41] T. Song, X. Liu, and X. Zeng, "Asynchronous spiking neural P systems with anti-spikes," *Neural Process. Lett.*, vol. 42, no. 3, pp. 633–647, Dec. 2015.

[42] T. Song, L. Pan, T. Wu, P. Zheng, M. L. D. Wong, and A. Rodríguez-Patón, "Spiking neural P systems with learning functions," *IEEE Trans. Nanobiosci.*, vol. 18, no. 2, pp. 176–190, Apr. 2019.

[43] J. Li, Y. Huang, and J. Xu, "Decoder design based on spiking neural P systems," *IEEE Trans. Nanobiosci.*, vol. 15, no. 7, pp. 639–644, Oct. 2016.
[44] S. Zhao, L. Zhang, Z. Liu, H. Peng, and J. Wang, "ConvSNP: A deep learning model embedded with SNP-like neurons," *J. Membrane Comput.*, vol. 4, no. 1, pp. 87–95, Mar. 2022.
[45] T. Song, X. Zeng, P. Zheng, M. Jiang, and A. Rodríguez-Patón, "A parallel workflow pattern modeling using spiking neural P systems with colored spikes," *IEEE Trans. Nanobiosci.*, vol. 17, no. 4, pp. 474–484, Oct. 2018.
[46] G. Rozenberg and A. Salomaa, *Handbook of Formal Language*. Berlin, Germany: Springer-Verlag, 1997.
[47] I. Korec, "Small universal register machines," *Theor. Comput. Sci.*, vol. 168, no. 2, pp. 267–301, 1996.
[48] H. Peng *et al.*, "Dendrite P systems," *Neural Netw.*, vol. 127, pp. 110–120, Jul. 2020.
[49] T. Bao, N. Zhou, H. Peng, Q. Yang, and J. Wang, "Computational completeness of sequential spiking neural P systems with inhibitory rules," *Inf. Comput.*, vol. 281, Dec. 2021, Art. no. 104786.
[50] Q. Ren and X. Liu, "Delayed spiking neural P systems with scheduled rules," *Complexity*, vol. 2021, pp. 1–13, Apr. 2021.
[51] T. Song, Y. Jiang, X. Shi, and X. Zeng, "Small universal spiking neural P systems with anti-spikes," *J. Comput. Theor. Nanosci.*, vol. 10, no. 4, pp. 999–1006, Apr. 2013.
[52] A. Leporati, C. Zandron, C. Ferretti, and G. Mauri, "On the computational power of spiking neural P systems," *Int. J. Unconv. Comput.*, vol. 5, no. 5, pp. 459–473, 2009.
[53] A. Leporati, G. Mauri, C. Zandron, G. Păun, and M. J. Pérez-Jiménez, "Uniform solutions to SAT and subset sum by spiking neural P systems," *Natural Comput.*, vol. 8, no. 4, pp. 681–702, Dec. 2009.
[54] F. G. C. Cabarle, R. T. A. de la Cruz, D. P. P. Cailipan, D. Zhang, X. Liu, and X. Zeng, "On solutions and representations of spiking neural P systems with rules on synapses," *Inf. Sci.*, vol. 501, pp. 30–49, Oct. 2019.
[55] T. Song, L. Luo, J. He, Z. Chen, and K. Zhang, "Solving subset sum problems by time-free spiking neural P systems," *Appl. Math. Inf. Sci.*, vol. 8, no. 1, pp. 327–332, Jan. 2014.
[56] F. G. C. Cabarle, N. H. S. Hernandez, and M. A. Martinez-del-Amor, "Spiking neural P systems with structural plasticity: Attacking the subset sum problem," presented at the 16th Int. Conf. Membrane Comput. (CMC), Valencia, Spain, Aug. 2015.
[57] T. Song and L. Pan, "Spiking neural P systems with request rules," *Neurocomputing*, vol. 193, no. 12, pp. 193–200, 2016.