Accepted Manuscript

International Journal of Neural Systems

Article Title:	Nonlinear Spiking Neural P Systems
Author(s):	Hong Peng, Zeqiong Lv, Bo Li, Xiaohui Luo, Jun Wang, Xiaoxiao Song, Tao Wang, Mario J. Perez-Jimenez, Agustin Riscos-Nunez
DOI:	10.1142/S0129065720500082
Received:	02 December 2019
Accepted:	03 December 2019
To be cited as:	Hong Peng *et al.*, Nonlinear Spiking Neural P Systems, *International Journal of Neural Systems*, doi: 10.1142/S0129065720500082
Link to final version:	https://doi.org/10.1142/S0129065720500082

International Journal of Neural Systems, Vol. 0, No. 0 (2015) 1–17

Nonlinear Spiking Neural P Systems

Hong Peng, Zeqiong Lv, Bo Li, and Xiaohui Luo
School of Computer and Software Engineering
Xihua University, Chengdu, 610039, China
ph.xhu@hotmail.com

Jun Wang,* Xiaoxiao Song, and Tao Wang
School of Electrical Engineering and Electronic Information
Xihua University, Chengdu, 610039, China
**wj.xhu@hotmail.com, songxx_xhu@163.com, wangatao2005@163.com*

Mario J. Pérez-Jiménez and Agustín Riscos-Núñez
Research Group of Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Sevilla, 41012, Spain
marper@us.es, ariscosn@us.es

This paper proposes a new variant of spiking neural P systems (in short, SNP systems), nonlinear spiking neural P systems (in short, NSNP systems). In NSNP systems, the state of each neuron is denoted by a real number, and a real configuration vector is used to characterize the state of whole system. A new type of spiking rules, nonlinear spiking rules, is introduced to handle the neuron's firing, where the consumed and generated amount of spikes are often expressed by the nonlinear function of the state of neuron. NSNP systems are a class of distributed parallel and non-deterministic computing systems. The computational power of NSNP systems is discussed. Specifically, it is proved that NSNP systems as number generating/accepting devices are Turing universal. Moreover, we establish two small universal NSNP systems for function computing and number generator, containing 117 neurons and 164 neurons, respectively.

Keywords: Membrane computing; spiking neural P systems; nonlinear spiking neural P systems; universality; register machines.

1. Introduction

In the field of membrane computing[1–4], spiking neural P systems (in short, SNP systems) introduced by Ionescu et al.[5] are a class of distributed parallel computing systems, abstracted by the way that the neurons deal with and communicate information to each other by sending spikes along synapses. Due to the inspiration of spiking neurons, spiking neural networks (SNNs)[6–16] and SNP systems are regarded as one of the 3rd generation artificial neural networks (ANNs), however, they have more differences in terms of working mechanism and dynamic behavior. This paper mainly focusses on SNP systems. From a topological perspective, SNP systems are expressed by a directed graph, where the neurons denote the nodes of the graph and the arcs are the synaptic connections between these neurons. Moreover, SNP systems have two important components: (i) data, and (ii) rules (to deal with the data). Different from other types of P systems (for example, cell-like P

*Corresponding author.

systems[17; 18], tissue-like P systems[19−22], population P systems[23; 24] and P colony[25]), only one type of object, known as the spike (denoted by symbol "a"), is considered in SNP systems. The state of each neuron is denoted by the number of spikes contained in it, and the state of the system is characterized by a configuration (vector) during the computation. The state in each neuron is evolved by the rules contained in it. There are rules of two types in SNP systems: spiking rule and forgetting rule. The spiking rule has the form $E/a^c \rightarrow a^p$. If spiking rule $E/a^c \rightarrow a^p$ is applied in a neuron, then c spikes are consumed and p spikes are produced and sent to its subsequent neurons. The forgetting rule has the form $a^s \rightarrow \lambda$, and if the rule is applied in a neuron, then s spikes are removed from the neuron and no spike is produced. Since two or more rules in a neuron can be applied at the same time, one of them is chosen non-deterministically. Therefore, non-determinism is an interesting feature of SNP systems. Moreover, SNP systems can work in three modes: generating, accepting and computing modes.

Abstracted by different biological mechanisms and/or instroduced the methods in mathematics or computer sciences, a lot of variants of SNP systems have been proposed in the past years. With this biological inspiration that astrocytes have excitatory and inhibitory influence on synapses, SNP systems with astrocytes have been proposed in Păun[26] and Pan et al.[27], respectively. Considering a pair of anti-spikes $(a, \bar{a})$, Pan et al.[28] discussed SNP system with anti-spikes. Abstrzcted by the biological fact that the synapse has one or more chemical channels, SNP systems with multiple channels were discussed in Peng et al.[29] and Song et al.[30], respectively. Considering that the rules in neurons are moved into synapses, SNP systems with rules on synapses have been investigated in Song et al.[31], and another spike consumption strategy was adopted in Peng et al.[32]. In Chen et al.[33], an axon P system was investigated, where the nodes were arranged in a linear structure and each of them only sends the spikes to two neighbors. Inspired from the biological phenomena that every neuron has a negative or positive charge, SNP systems with polarizations were investigated in Wu et al.[34]. Abstracted by the structural dynamism of biological synapses, Cabarle et al.[35] presented an SNP system with scheduled synapses. Considering a new communication strategy among neurons, Pan et al.[36] investigated an SNP systems with communication on request. With the restriction that at each step one neuron works at most, several sequential SNP systems were discussed in Ibarra et al.[37] and Zhang et al.[38], respectively. Wang et al.[39] discussed an SNP system with weights, where weights were introduced on synapses similar to those in artificial neural networks. Dynamic threshold neural P systems and coupled neural P systems were discussed in Peng et al.[40; 41], respectively. Moreover, using the thresholds instead of the original regular expression, Zeng et al. [42] proposed SNP systems with thresholds. Note that in SNP systems with thresholds, the firing condition is changed as $n \geq T$. Usually, a global clock is assumed in SNP system, so they are synchronized. However, two asynchronous SNP systems were investigated in Cavaliere et al.[43] and Song et al.[44]. By integrating fuzzy logics in SNP systems, several fuzzy SNP systems were developed in the recent years, for example, fuzzy reasoning SNP systems[45], weighted fuzzy SNP systems[46] and Interval-valued fuzzy SNP systems[47]. Computational power of SNP systems has been investigated: most variants of SNP systems were proven to be Turing universal as number generating/accepting devices[48; 49], language generating devices[50] and function computing devices[51]. SNP systems were used to (theoretically) solve computationally hard problems in a feasible (polynomial or linear) time[52; 53]. SNP systems have been applied to solve some real-world problems, such as fault diagnosis[54−56], image processing[57; 58] and combinatorial optimization problems[59].

As mentioned above, the existing SNP systems and variants have two features:

(i) The state of each neuron is an integer, denoted by the number of spikes;
(ii) It can be observed from spiking rule $E/a^c \rightarrow a^p$ that the consumed and generated numbers of spikes are two constants, c and p, and are independent on the state of the neuron in which the rule resides. As a result, its state equation is a linear equation, of the form $n(t) = n(t-1) - c + p'$, where $n(t)$ denotes the current state of the neuron, p is the number of spikes consumed by the rule and p' is the sum of spikes received from its precursor neurons.

In this paper, a new variant of SNP systems is investigated, nonlinear spiking neural P systems (in short, NSNP systems). Compared with SNP systems and the existing variants, NSNP systems have three new characteristics: (i) the state of each neuron is denoted by a real number; (ii) the consumed and generated amount of spikes in spiking rules are expressed by two nonlinear functions of the state of the system; (iii) nonlinear firing rules are introduced to process the neuron's firing. This paper focusses on computational power of NSNP systems. Turing universality of NSNP systems as number generating/accepting devices is first investigated. Then, two small universal NSNP systems, respectively, as function computing and number generating devices, are constructed.

The remainder of this paper is organized as follows. In Section 2, we review some mathematical preliminaries that will be used in the following. Section 3 first reviews original SNP systems, and then describes in detail the proposed NSNP systems. In Section 4, universality of NSNP systems as number generating/accepting devices is discussed. Section 5 constructs a small universal function computing device and a small number generating devices. Conclusions and future work is drawn in Section 6.

2. Prerequisites

We assume the reader to be familiar with basic elements about SNP systems, automata and language theory. Some notations and notions of register machines, used later in the proofs of our results, are introduced here.

For an alphabet O, denote by O^* the set of all finite strings over O and by O^+ the set of all nonempty strings over O, and the empty string is denoted by λ. If $O = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$.

A regular expression over O is defined as follows: (i) λ and each $a \in O$ is a regular expression, (ii) if E_1, E_2 are regular expressions over O, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over O, and (iii) nothing else is a regular expression over O. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in O$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over O. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

A register machine is defined by a tuple $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H denotes the set of instruction labels, l_0 is the starting label, l_h is the halting instruction $HALT$, and I denotes the set of instructions. Each label in H is associated with an instruction in I. Each instruction in I is one of the following three forms:

(1) $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then go non-deterministically to one of instructions with labels l_j, l_k).
(2) $l_i : (SUB(r), l_j, l_k)$ (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise go to the instruction with label l_k).
(3) $l_h : HALT$ (the halting instruction).

Denote by $N_{gen}(M)$ the set of numbers that are non-deterministically generated by register machine M. Its working mechanism can be explained as follows. At the beginning, all registers except the first register are empty, and the first register is not affected by SUB instructions during the computation. Register machine M starts from instruction l_0 and then executes constantly other instructions in I. When it turns to instruction l_h, the computation is accomplished, and the final result is stored in the first register.

The register machine M can be used to accept numbers, i.e., it works in accepting mode. Denote by $N_{acc}(M)$ the set of numbers that are accepted by register machine M. Its working mechanism can be explained as follows. At the beginning, all registers are empty except the first register, and a number is introduced into the first register. Starting from instruction l_0, register machine M executes constantly the instructions in I until the halting instruction l_h arrives. Thus, the number is called to be accepted by register machine M. In accepting mode, register machine is deterministic, meaning that $l_i : (ADD(r), l_j)$ instead of $l_i : (ADD(r), l_j, l_k)$ is used as the ADD instruction.

It is well-known that a register machine M can generate/accept any Turing computable number set (denoted by NRE). That is, register machine M working on generating/accepting mode is Turing universal. Therefore, register machine will be used as a standard model for universality investigation of the variant proposed in this paper.

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

Fig. 1. The small universal register machine M'_u.

A register machine can be used for computing Turing-computable function $f : N^k \to N$. Its working principle can be explained as follows. Initially, k arguments are introduced into k special registers (usually, the first k registers are used), and all other registers are assumed to be empty. Starting from instruction l_0, the register machine is executed constantly until it reaches halting instruction l_h. At this moment, the value of function f is the number stored in another special register r_t. Denote by $(\varphi_0, \varphi_1, \ldots)$ a fixed admissible enumeration of the unary partial recursive functions. A register machine is said to be universal if only if there exists a recursive function g so that for all natural numbers x, y, $\varphi_x(y) = M_u(g(x), y)$ holds.

In Korec[60], a small universal register machine for computing functions was introduced, $M_u = (8, H, l_0, l_h, I)$. The register machine M_u contains 23 instructions and 8 registers (labeled by digits from 0 to 7). By introducing two numbers $g(x)$ and y in registers 1 and 2 respectively, any $\varphi_x(y)$ can be computed by register machine M_u; the function value is stored in register 0 when the register machine halts.

For simplicity, register machine M_u is modified: a new register 8 is introduced, and original halting instruction is substituted by the following three instructions: $l_{22} : (SUB(0), l_{23}, l_h); l_{23} : (ADD(8), l_{22}); l_h : HALT$. Fig. 1 shows the modification of M_u, denote by M'_u. Therefore, register machine M'_u contains 25 instructions (10 ADD instructions, 14 SUB instructions and 1 HALT instruction), 9 registers and 25 labels.

In this paper, register machine M'_u will be used as a standard model for the discussion of universality of the proposed variant as a function computing device.

3. Nonlinear Spiking Neural P Systems

In this section, we first review the definition and mechanism of SNP systems, and then introduce the definition of NSNP systems and discuss their working mechanism.

3.1. *SNP systems*

Definition 3.1. An SNP system, of degree $m \geq 1$, is a tuple:

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

where

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

(2) $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i=(n_i, R_i)$, $1 \leq i \leq m$, where

  (a) $n_i \geq 0$ is the initial number of spikes contained in neuron σ_i;

  (b) R_i is the finite set of spiking rules of the form $E/a^c \to a^p$, where E is a regular expression over O, and $c \geq p \geq 0$.

(3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$ (synapses).

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

An SNP system can be expressed by a directed graph, where m neurons, $\sigma_1, \ldots, \sigma_m$, are the nodes of the graph and the arcs denote the synaptic connections between these neurons. Each neuron has a data unit that is used to denote the number of spikes stored in it, which describes the state of the neuron during the computation. The state of each neuron is evolved by its rules. There are two types of rules: spiking rule and forgetting rule. The spiking rule is with the form $E/a^c \to a^p$, and the corresponding firing condition is $a^n \in L(E)$, where n denotes the number of spikes stored in the neuron that the rule resides and $L(E)$ denotes the language generated by E. If spiking rule $E/a^c \to a^p$ is applied, then c spikes are consumed and p spikes are produced and sent to its subsequent neurons. When $p = 0$, the rule is known as forgetting rule, written as $a^c \to \lambda$, where symbol λ denotes the empty string. The corresponding firing condition is that the neuron contains precisely c spikes. If the forgetting rule is used, then c spikes are removed from the neuron and no spike is produced.

Based on firing semantics of spiking rules and forgetting rules above, state equation of neuron σ_i can be expressed as follows:

$$n_i(t+1) = \begin{cases} n_i(t) - c + p', & \text{if neuron } \sigma_i \text{ fires} \\ n_i(t) + p', & \text{otherwise} \end{cases} \tag{1}$$

where $n_i(t+1)$ and $n_i(t)$ are the states of neuron σ_i at time $t+1$ and time t respectively, that is, the numbers of spikes stored in neuron σ_i at the two moments; c is the number of spikes consumed by spiking rule or forgetting rule; p' is the number of spikes received by neuron σ_i from its predecessor neurons. It is important to point out that the state equation (1) is essentially a linear equation because c and p are two constants and are independent of the current state of the system.

3.2. *NSNP systems*

Definition 3.2. An NSNP system, of degree $m \geq 1$, is a tuple:

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$$

where

(1) $O=\{a\}$ is the singleton alphabet (a is known as the spike);
(2) $\sigma_1, \dots, \sigma_m$ are m neurons, of the form

$$\sigma_i = (x_i, R_i),\ 1 \leq i \leq m$$

where

(a) $x_i \in R^+$ is initial value of spikes contained in neuron σ_i, indicating initial state of neuron σ_i;
(b) R_i is the finite set of spiking rules, of the form $T|a^{g(x_i)} \to a^{f(x_i)}$, where $T \in R^+$ is a firing threshold, $g(x_i)$ is a linear or nonlinear function and $f(x_i)$ is a nonlinear function, and $g(x_i) \geq f(x_i) \geq 0$.

(3) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$ (synapses).
(4) in, out indicate the input and output neurons of the system, respectively.

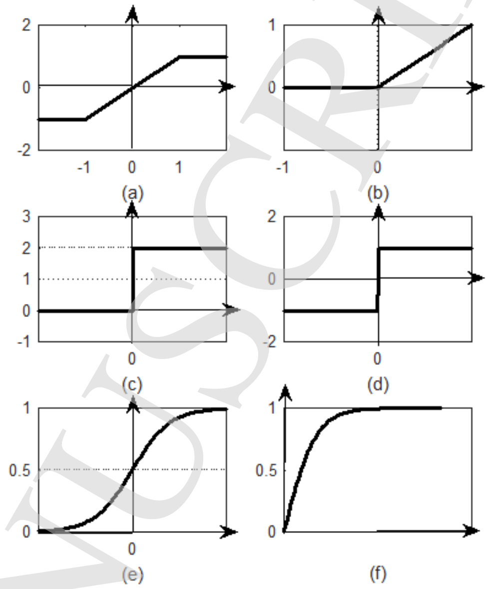

Fig. 2. Six simple nonlinear functions: (a) $f_1(x)$, (b) $f_2(x)$, (c) $f_3(x)$, (d) $f_4(x)$, (e) $f_5(x)$ and (f) $f_6(x)$.

As usual, an NSNP system can be also expressed by a directed graph, where the m neurons are the nodes and the arcs denote the synaptic connections between these neurons. Specifically, if NSNP system works in generative mode, then input neuron σ_{in} is omitted, however, in accepting mode, output neuron σ_{out} is removed from the system.

In comparison with SNP systems, there are three differences in NSNP systems:

(1) The state of each neuron is changed as a real number in R^+, indicating the value of spikes contained in the neuron, where R^+ denotes the set of nonnegative real numbers.
(2) The threshold firing condition is used, i.e., $x_i \geq T$, where $T \in R^+$ is a threshold constant.
(3) The nonlinear spiking rules, $T/a^{g(x_i)} \to a^{f(x_i)}$, are introduced, where $g(x_i)$ is a linear or nonlinear function (called consumption function), and $f(x_i)$ is a nonlinear function (called generation function).

Fig. 2 shows six simple nonlinear functions that can be used as consumption function and generation function in nonlinear spiking rules. The six nonlinear functions are defined as follows.

(1) $f_1(x) = \begin{cases} 1, x > 1 \\ x, -1 \leq x \leq 1 \\ -1, x < -1 \end{cases}$

(2) $f_2(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$

(3) $f_3(x) = \begin{cases} 2, x > 0 \\ 0, x \leq 0 \end{cases}$

(4) $f_4(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$

(5) Logistic function:

$$f_5(x) = \frac{1}{1 + e^{-cx}}$$

(6) Sigmoid function:

$$f_6(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

In NSNP systems, nonlinear spiking rules with the form $T|a^{g(x_i)} \to a^{f(x_i)}$ are applied to evolve the states in neurons. Note that consumption function $g(x_i)$ is a linear or nonlinear function of state x_i of neuron σ_i, and generation function $f(x_i)$ is a nonlinear function of state x_i of neuron σ_i. The firing semantics of neurons can be explained as follows. Assume that $x_i(t)$ denotes the state of neuron σ_i at time t. The firing condition is $x_i(t) \geq T$. When $x_i(t) \geq g(x_i)$ and $x_i(t) \geq T$ are satisfied, spiking rule $T|a^{g(x_i)} \to a^{f(x_i)}$ can be applied. If the spiking rule in neuron σ_i is applied, then it fires and the spikes with value $g(x_i(t))$ are consumed (the spikes with value $(x_i(t) - g(x_i(t)))$ are retained in the neuron), and then the spikes with value $f(x_i(t))$ are generated. The generated spikes with value $f(x_i(t))$ are sent to neurons $\{\sigma_j|(i,j) \in syn\}$. If $f(x_i(t)) \equiv 0$, spiking rule is known as forgetting rule, written as $T|a^{g(x_i)} \to \lambda$. Applying the forgetting rule means that the spikes with value $g(x_i(t))$ are removed from the neuron and no spike is generated.

Based on firing semantics of spiking rules and forgetting rules above, state equation of neuron σ_i can be expressed as follows:

$$x_i(t+1) = \begin{cases} x_i(t) - g(x_i(t)) + y_i(t), & \text{if neuron } \sigma_i \text{ fires} \\ x_i(t) + y_i(t), & \text{otherwise} \end{cases} \quad (2)$$

where $x_i(t+1)$ and $x_i(t)$ are the states of neuron σ_i at time $t+1$ and time t respectively, i.e., the values of spikes contained in neuron σ_i at the two moments; $g(x_i(t))$ is the value of spikes consumed by spiking rule or forgetting rule; $y_i(t)$ is the value of spikes received by neuron σ_i from its predecessor neurons. Since $g(x_i(t))$ and $y_i(t)$ are linear or nonlinear function of state $x_i(t)$ of neuron σ_i, stare equation (2) is a nonlinear equation.

Since two rules in neuron σ_i, rule $T_1|a^{g_1(x_i)} \to a^{f_1(x_i)}$ and rule $T_2|a^{g_2(x_i)} \to a^{f_2(x_i)}$, may satisfy $\{x_i \geq T_1\} \cap \{x_i \geq T_2\} \neq \emptyset$. At this time, two cases are considered as follows:

(1) If $T_1 \neq T_2$, maximum threshold strategy is adapted. For example, if $T_1 > T_2$, then rule $T_1|a^{g_1(x_i)} \to a^{f_1(x_i)}$ is chosen and applied, however, rule $T_2|a^{g_2(x_i)} \to a^{f_2(x_i)}$ can not be used. Note that one of the two rules may be a forgetting rule. For example, if rule $T_1|a^{g_1(x_i)} \to \lambda$ and rule $T_2|a^{g_2(x_i)} \to a^{f_2(x_i)}$ can be applied, rule $T_1|a^{g_1(x_i)} \to \lambda$ is applied but rule $T_2|a^{g_2(x_i)} \to a^{f_2(x_i)}$ is not used.

(2) If $T_1 = T_2 = T$, non-deterministic rule selection strategy is used. If rule $T|a^{g_1(x_i)} \to a^{f_1(x_i)}$ and rule $T|a^{g_2(x_i)} \to a^{f_2(x_i)}$ can be applied, one of them is chosen non-deterministically.

As in SNP systems, the rules are used in the sequential manner in each neuron, but neurons work in parallel. Therefore, NSNP systems are a distributed parallel and non-deterministic computing systems.

The configuration of the system at time t can be described by the state of each neuron, i.e., $X(t) = (x_1(t), x_2(t), \ldots, x_m(t))$. Thus, initial configuration is denoted by $X(0) = (x_1(0), x_2(0), \ldots, x_m(0)) = (x_1, x_2, \ldots, x_m)$. By applying spiking rules and forgetting rules, one can define a transition from one configuration to another. Any sequence of transitions starting from the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied.

4. Computational Completeness

In this section, we will discuss Turing universality of NSNP systems as number generating/accepting devices. Since register machines are universal number generating/accepting devices, the universality of N-SNP systems will be proven by simulating register machines. By means of register machines, all recursively enumerable sets of numbers (the family is denoted by NRE) can be generated/accepted by NSNP systems.

It is important to point out that the state of neuron is often restricted to nonnegative integer when discussing the universality. Therefore, integer value of spikes contained in each neuron can be understood as the number of spikes as in SNP systems. For this purpose, two nonlinear functions are considered:

$$\beta(x) = \begin{cases} x, \, x > 0 \\ 0, \, x \leq 0 \end{cases}, \gamma(x) = \begin{cases} 2, \, x > 0 \\ 0, \, x \leq 0 \end{cases}$$

For the convenience to describe, in the discussion of universality later, if the state of a neuron is a nonnegative integer $n \geq 0$, it is said that the neuron has n spikes.

In the discussion of universality, we will design some modules of NSNP systems to simulate the corresponding instructions of register machines. In these modules, the following assumptions are given:

(i) Each instruction l in H is associated with a neuron σ_l, such as instruction $l_i \Leftrightarrow$ neuron σ_{l_i}, instruction $l_j \Leftrightarrow$ neuron σ_{l_j} and instruction $l_k \Leftrightarrow$ neuron σ_{l_k}. Moreover, some auxiliary neurons are considered in these modules, for example, $\sigma_{c_1}, \sigma_{c_2}$.

(ii) Each register r in register machine is associated with a neuron σ_r, for example, register $r \Leftrightarrow$ neuron σ_r, register $1 \Leftrightarrow$ neuron σ_1 and register $2 \Leftrightarrow$ neuron σ_2.

(iii) The number in register r is coded: if register r contains the number $n \geq 0$, then the corresponding neuron σ_r has $2n$ spikes, and vice versa. That is, number n in register $r \Leftrightarrow 2n$ spikes in neuron σ_r.

(iv) if an instruction neuron σ_l (for example, σ_{l_i} or σ_{l_j} or σ_{l_k}) receives two spikes, then it is activated, meaning that the corresponding instruction l (for example, l_i or l_j or l_k) begins to be simulated.

As usual in SNP systems, we can define the result of a computation as the number of steps between the first two spikes sent out by the output neuron. Denote by $N_2(\Pi)$ the set of numbers generated by Π. Denote by $N_2NSNP_m^n$ the family of all sets $N_2(\Pi)$ generated by NSNP systems having at most m neurons and at most n rules in each neuron.

NSNP systems can also work in accepting mode, where the output neuron is omitted, but an input neuron receives a spike train from the environment. The system starts by reading the spike train from the environment, and then a number n is introduced in a specified neuron in the form of $2n$ spikes. If the system can halt, then it is said that it can accept the number n. Denoted by $N_{acc}(\Pi)$ the set of numbers accepted by system Π, where *acc* indicates that the system works in accepting mode. Denote by $N_{acc}NSNP_m^n$ the family of all sets $N_{acc}(\Pi)$ accepted by NSNP systems containing at most m neurons and at most n rules in each neuron.

4.1. *NSNP systems as number generating device*

In generating mode, a register machine can compute a number n in the following way: with all empty registers, starting from the instruction l_0, the machine continuously applies the instructions as indicated by labels; if it reaches the halting instruction l_h, then the number contained in the first register is said to be computed by register machine M. It is well-known that the family NRE can be characterized by register machines. We provide the universal result of N-SNP systems working in generating mode as number generating device as follows.

Theorem 4.1. $N_2NSNP_*^2 = NRE$

Proof. Because $N_2NSNP_*^2 \subseteq NRE$ is straightforward, only inclusion $NRE \subseteq N_2NSNP_*^2$ needs to be proved. We will use the NRE characterized by register machine in generating mode. Let $M_1 = (m, H, l_0, l_h, I)$ be the register machine. Generally, suppose that in the halting configuration, all registers different from register 1 are empty, and register 1 is never decremented during the computation.

For simulating register machine M_1, we construct an NSNP system Π_1, including modules of three types: ADD module (simulating ADD instruction of M_1, shown in Fig. 3), SUB module (simulating SUB instruction of M_1, shown in Fig. 4), and a FIN module (outputting the computation result) shown in Fig. 5.

Initially, no spike is stored in all auxiliary neurons, and during the computation neuron σ_{l_i} that are associated with instruction l_i receives two spikes (i.e., the spikes with value of 2, however, for simple, it is said to be two spikes, the same below). With two spikes in neuron σ_{l_i}, system Π_1 starts the simulation of an instruction $l_i : (OP(r), l_j, l_k)$ (OP is one of operations ADD and SUB): the simulation starts from the activated neuron σ_{l_i} and handles neuron

σ_r as labeled by OP, and then sends two spikes in one of neurons σ_{l_j} and σ_{l_k}. The system finishes the simulation once neuron σ_{l_h} is activated. During the computation, output neuron σ_{out} applies its rules to send the spikes to the environment twice at times t_1 and t_2 respectively, and the time interval $t_2 - t_1$ that is associated with the number contained in register 1 in M_1 is regarded as the computation result.

To prove that register machine M_1 can be correctly simulated by system Π_1, it will be illustrated how ADD and SUB modules simulate the ADD and SUB instructions respectively, and how the FIN module outputs the computation result.

(1) ADD Module (shown in Fig. 3) - simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$.

The system Π_1 starts from the simulation of instruction l_0, which is an ADD instruction. Generality, assume that an ADD instruction l_i : $(ADD(r), l_j, l_k)$ is simulated at time t. With two spikes in neuron σ_{l_i} (its state is $x = 2$), rule $2|a^x \to a^{\beta(x)}$ is used, and then two spikes are consumed and two spikes are generated and sent to neurons σ_{c_1} and σ_r (because of $\beta(x) = 2$). Thus, neuron σ_r receives two spikes, meaning that register r is added by 1.

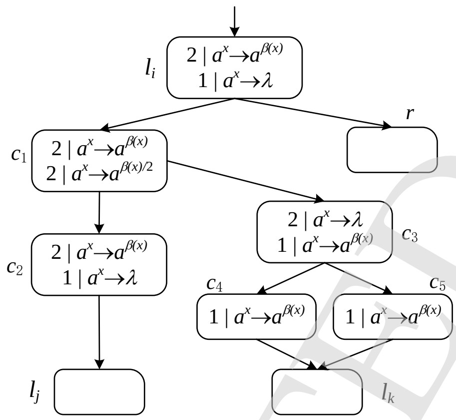

Fig. 3. ADD module (simulating $l_i : (ADD(r), l_j, l_k)$).

At time $t+1$, with two spikes in neuron σ_{c_1}, rule $2|a^x \to a^{\beta(x)}$ and rule $2|a^x \to a^{\beta(x)/2}$ can be applied. Since the two rules have the same threshold, one of them is selected non-deterministically. There are the following two cases:

(i) At time $t+1$, if rule $2|a^x \to a^{\beta(x)}$ is applied, neuron σ_{c_1} sends two spikes to neurons σ_{c_2} and σ_{c_3} respectively (because of $x = 2$ and $\beta(x) = 2$). At time $t+2$, with two spikes in neuron σ_{c_2}, rule $2|a^x \to a^{\beta(x)}$ is applied, then two spikes are sent to neuron σ_{l_j} (because of $x = 2$ and $\beta(x) = 2$). Thus, neuron σ_{l_j} receives two spikes, meaning that system Π_1 starts the simulation of instruction l_j of M_1. At the same time, since rule $2|a^x \to \lambda$ in neuron σ_{c_3} can be applied, its two spikes are removed by the forgetting rule.

(ii) At time $t+1$, if rule $2|a^x \to a^{\beta(x)/2}$ is applied, neuron σ_{c_1} sends a spike to neurons σ_{c_2} and σ_{c_3} respectively (because of $x = 2$ and $\beta(x)/2 = 1$). At time $t+2$, with a spike in neuron σ_{c_3}, rule $1|a^x \to a^{\beta(x)}$ is applied, then a spike is sent to neurons σ_{c_4} and σ_{c_5} (because of $x = 1$ and $\beta(x) = 1$). However, since neuron σ_{c_2} has only one spike, the spike is removed by $1|a^x \to \lambda$. At time $t+3$, neurons σ_{c_4} and σ_{c_5} each send a spike to neuron σ_{l_k} by rule $1|a^x \to a^{\beta(x)}$ (because of $x = 1$ and $\beta(x) = 1$). Thus, neuron σ_{l_k} receives two spikes in total. With two spikes in neuron σ_{l_k}, system Π_1 starts the simulation of instruction l_k of M_1.

Consequently, ADD module can correctly simulate ADD instruction: from neuron σ_{l_i} receiving two spikes, value of spikes in neuron σ_r is added by two, and one of neurons σ_{l_j} and σ_{l_k} is non-deterministically chosen.

(2) SUB module (shown in Fig. 4) - simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$.

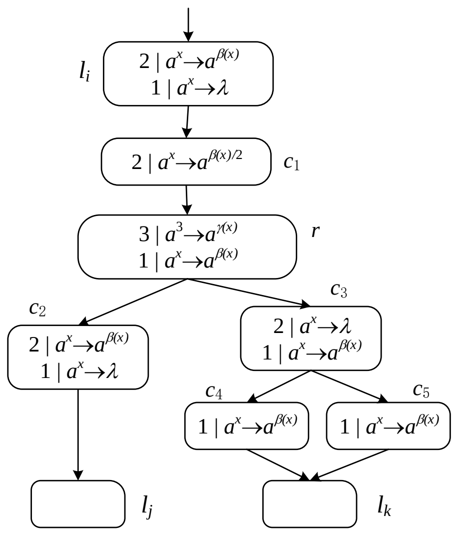

Fig. 4. SUB module (simulating $l_i : (SUB(r), l_j, l_k)$).

Assume that a SUB instruction l_i : $(SUB(r), l_j, l_k)$ is simulated at time t, and neuron σ_{l_i} receives two spikes. With two spikes in neuron σ_{l_i} (meaning its state $x = 2$), rule $2|a^x \rightarrow a^{\beta(x)}$ is used, and it sends two spikes to neuron σ_{c_1} (because of $\beta(x) = 2$). At time $t+1$, with two spikes in neuron σ_{c_1}, it applies rule $2|a^x \rightarrow a^{\beta(x)/2}$ to send a spike to neuron σ_r (because of $\beta(x)/2 = 1$). Thus, neuron σ_r receives a spike. According to the value of spikes in neuron σ_r, there are the following two cases:

(i) At time $t+2$, if neuron σ_r contains $2n+1$ spikes (i.e., its state is $x = (2n+1) \geq 3$, corresponding to the fact that the number contained in register r is n), then rule $3|a^3 \rightarrow a^{\gamma(x)}$ is applied, and then two spikes are sent to neurons σ_{c_2} and σ_{c_3} (because of $\gamma(x) = 2$ when $x \geq 0$), respectively. At time $t+3$, with two spikes in neuron σ_{c_2}, rule $2|a^x \rightarrow a^{\beta(x)}$ is applied, and two spikes are sent to neuron σ_{l_j} (because of $x = 2$ and $\beta(x) = 2$). Thus, neuron σ_{l_j} receives two spikes, meaning that system Π_1 starts the simulation of instruction l_j of M_1. At the same time, since rule $2|a^x \rightarrow \lambda$ in neuron σ_{c_3} can be applied, the received two spikes are removed by the forgetting rule.

(ii) At time $t+2$, if neuron c_r has only one spike (i.e., its state is $x = 1$, corresponding to the fact that the number stored in register r is zero), then rule $1|a^x \rightarrow a^{\beta(x)}$ is applied and a spike is sent to neurons σ_{c_2} and σ_{c_3} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t+3$, with a spike in neuron σ_{c_3}, rule $1|a^x \rightarrow a^{\beta(x)}$ is used and a spike is sent to neurons σ_{c_4} and σ_{c_5} respectively (because of $x = 1$ and $\beta(x) = 1$). However, since neuron σ_{c_2} has only one spike, the spike is removed by rule $1|a^x \rightarrow \lambda$. At time $t+4$, neurons σ_{c_4} and σ_{c_5} each send a spike to neuron σ_{l_k} by rule $1|a^x \rightarrow a^{\beta(x)}$ respectively (because of $x = 1$ and $\beta(x) = 1$). Thus, neuron σ_{l_k} receives two spikes. With two spikes in neuron σ_{l_k}, system Π_1 starts the simulation of instruction l_k of M_1.

Consequently, SUB instruction is correctly simulated by SUB module: the system starts from neuron σ_{l_i} receiving two spikes, and ends with sending two spikes to neuron σ_{l_j} (if the number contained in register r is greater than 0), or sending two spikes to neuron σ_{l_k} (if the number contained in register r is 0).

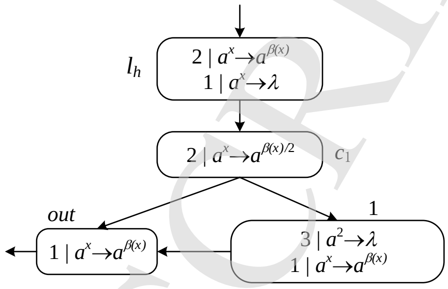

Fig. 5. FIN Module (ending the computation).

(3) FIN module (shown in Fig. 5) - outputting the result of computation.

Suppose that neuron σ_{l_h} receives two spikes at time t, meaning that M_1 halts (the halting instruction l_h is reached), and neuron σ_1 contains $2n$ spikes (indicating that register 1 of M_1 has the number n). With two spikes in neuron σ_{l_h}, rule $2|a^x \rightarrow a^{\beta(x)}$ is used to send a spike to neuron σ_{c_1} (because of $x = 2$ and $\beta(x) = 2$). At time $t+1$, with two spikes in neuron σ_{c_1}, rule $2|a^x \rightarrow a^{\beta(x)/2}$ is used to send a spike to neurons σ_1 and σ_{out} respectively (because of $x = 2$ and $\beta(x)/2 = 1$).

At time $t+2$, with a spike in neuron σ_{out}, rule $1|a^x \rightarrow a^{\beta(x)}$ is used to send the first spike to the environment (because of $x = 1$ and $\beta(x) = 1$). Since neuron σ_1 receives a spike, the value of spikes in it becomes odd, hence, rule $3|a^2 \rightarrow \lambda$ or rule $1|a^x \rightarrow a^{\beta(x)}$ can be applied. If neuron σ_1 contains $(2n+1) \geq 3$ spikes, then rule $3|a^2 \rightarrow \lambda$ is applied based on maximum threshold strategy. Thus, from time $t+2$ to time $t+n$, two spikes are removed in neuron σ_1 by the forgetting rule each time.

At time $t+n+1$, neuron σ_1 contains only one spike, hence, rule $1|a^x \rightarrow a^{\beta(x)}$ is used to send a spike to neuron σ_{out} (because of $x = 1$ and $\beta(x) = 1$). At $t+n+2$, with a spike in neuron σ_{out}, rule $1|a^x \rightarrow a^{\beta(x)}$ is used to send the second spike to the environment (because of $x = 1$ and $\beta(x) = 1$). Consequently, the time interval between the two spikes sent to the environment by the system is $(t+n+2)-(t+2) = n$, which is exactly the number contained in register 1 when M_1 halts.

From the above description of the modules of system Π_1, it is clear that system Π_1 correctly simulates register machine M_1, in which each neuron has at most two rules. Therefore, the theorem holds. $\square$

4.2. *NSNP systems as number accepting device*

We consider NSNP systems working in accepting mode as number accepting device, the corresponding universal result is given as follows.

Theorem 4.2. $N_{acc}NSNP^2_* = NRE$

Proof. We construct an NSNP system Π_2 for simulating a deterministic register machine $M_2 = (m, H, l_0, l_h, I)$ working in accepting mode. The following proof is described by modifying the proof of Theorem 4.1. The system Π_2 contains an INPUT module, a deterministic ADD module and a SUB module. Initially, no spike is contained in all auxiliary neurons of these modules.

Fig. 6 shows the INPUT module. Neuron σ_{in} reads a spike train $10^{n-1}1$, where the interval between the two spikes in the spike train is $(n+1)-1 = n$, meaning that the number to be accepted is n.

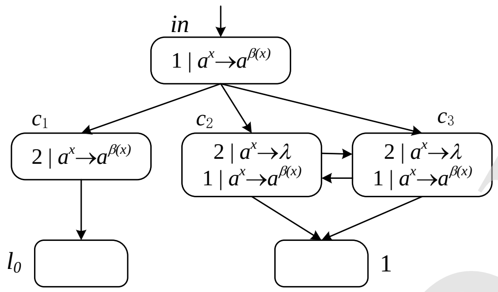

Fig. 6. INPUT Module of Π_2.

Suppose that at time t, neuron σ_{in} receives the first spike from the environment. With a spike in neuron σ_{in}, rule $1|a^x \to a^{\beta(x)}$ is applied to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t + 1$, neurons σ_{c_2} and σ_{c_3} each apply rule $1|a^x \to a^{\beta(x)}$ to send a spike to each other and to neuron σ_1 (because of $x = 1$ and $\beta(x) = 1$). Thus, neuron σ_1 receives two spikes. Since neuron σ_{c_1} has only one spike, rule $2|a^x \to a^{\beta(x)}$ can not be used.

From time $t + 1$ until neuron σ_{in} receives the second spike, neurons σ_{c_2} and σ_{c_3} each send a spike to each other and to neuron σ_1. Therefore, from time $t + 2$ to time $t + n + 1$, neuron σ_1 receives $2n$ spikes in total (meaning that the number stored in register 1 is n).

At time $t+n+1$, neuron σ_{in} receives the second spike, and then it fires and applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t+n+2$, with two spikes in neuron σ_{c_1}, rule $2|a^x \to a^{\beta(x)}$ is used to send two spikes to neuron σ_{l_0} (because of $x = 2$ and $\beta(x) = 2$). With two spikes in neuron σ_{l_0}, the system starts the simulation of initial instruction l_0.

When a register machine works in accepting mode, a deterministic ADD instruction, $l_i : (ADD(r), l_j)$, is considered, shown in Fig. 7. Since neuron σ_{l_i} receives two spikes, rule $2|a^x \to a^{\beta(x)}$ is used to send two spikes to neurons σ_{l_j} and σ_r respectively (because of $x = 2$ and $\beta(x) = 2$). Since neuron σ_{l_j} receives two spikes, the system starts the simulation of instruction l_j. Moreover, neuron σ_r receives two spikes, meaning that the number contained in register r is increased by 1.

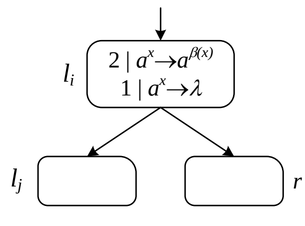

Fig. 7. Module ADD of Π' (simulating $l_i : (ADD(r), l_j)$).

SUB module remains unchanged (shown in Fig. 3). FIN module is removed, with neuron σ_{l_h} remaining in the system. When neuron σ_{l_h} receives two spikes, this indicates that the computation of register machine M_2 reaches instruction l_h and stops.

From the above description, we can find that the register machine working in accepting mode can be correctly simulated by NSNP system where each neuron has at most two rules. Hence, the theorem holds. □

5. Small Universal Computing Devices

In this section, we will investigate small universal function computing device and small universal number generating device (number generator). The universality of NSNP systems as function computing device means that NSNP systems can compute any Turing-computable function $f : N^k \to N$. An NSNP system is a computing system that consists of many neurons, where neurons are its computing units. Therefore, investigation of small universal devices is to design a universal NSNP system with the fewest number of neurons. For the purpose, small universal register machine M'_u is used as a standard model to prove that NSNP systems can be used as small universal function computing device and small universal number generating device.

5.1. *Small universal NSNP systems as function computing device*

Theorem 5.1. *There exists a small universal NSNP system having 117 neurons for computing functions.*

Proof. We construct an NSNP system Π_3 for the simulation of the universal register machine M'_u. The NSNP system Π_3 includes an INPUT module, an OUTPUT module and several ADD and SUB modules. The ADD and SUB modules are used to simulate ADD and SUB instructions of M'_u, respectively. The INPUT module reads a spike train from the environment, while the OUTPUT module outputs the computing result.

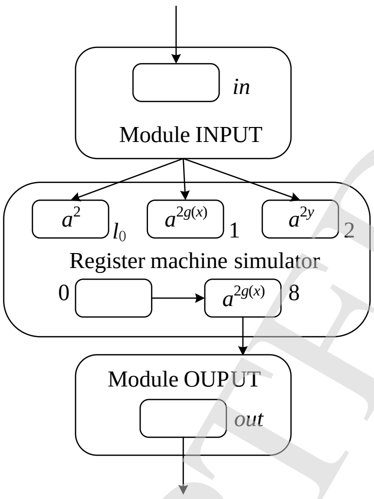

Fig. 8. The general design of the universal NSNP system Π_3.

The general design of the universal NSNP system Π_3 is provided in Fig. 8. Each register r in M'_u is associated with a neuron σ_r, and if register r stores the number $n \geq 0$, then neuron σ_r has $2n$ spikes. Moreover, neuron σ_{l_i} in Π_3 corresponds to instruction l_i in M'_u. When neuron σ_{l_i} receives two spikes, it starts the simulation of the instruction l_i. Once neuron σ_{l_h} receives two spikes, the system Π_3 completes the simulation of M'_u. Finally, the first two spikes sent to the environment by neuron σ_{out} are regarded as the computation result (stored in register 8). In the initial configuration, assume all neurons are empty.

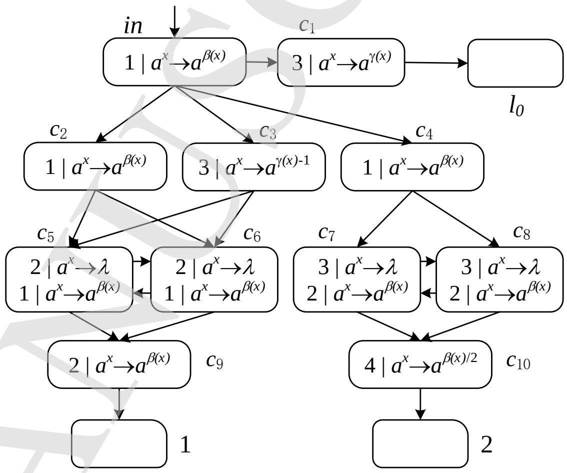

Fig. 9. INPUT module.

The INPUT module is shown in Fig. 9. The module reads a spike train $10^{g(x)-1}10^{y-1}1$ from the environment, where $2g(x)$ spikes are stored in neuron σ_1 and $2y$ spikes are placed in neuron σ_2.

Suppose that at time t_1, neuron σ_{in} receives the first spike from the environment. With a spike in neuron σ_{in}, rule $1|a^x \to a^{\beta(x)}$ is applied to send a spike to neurons σ_{c_1}, σ_{c_2}, σ_{c_3} and σ_{c_4} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t_1 + 1$, neuron σ_{c_2} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_5} and σ_{c_6} (because of $x = 1$ and $\beta(x) = 1$). At the same time, neuron σ_{c_4} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_7} and σ_{c_8} (because of $x = 1$ and $\beta(x) = 1$). However, neurons σ_{c_1} and σ_{c_3} can not be applied. At time $t_1 + 2$, neurons σ_{c_5} and σ_{c_6} each apply rule $1|a^x \to a^{\beta(x)}$ to a spike to each other, and at the same time they send a spike to neuron σ_{c_9} respectively (because of $x = 1$ and $\beta(x) = 1$). Thus, neuron σ_{c_9} receives two spikes. Despite neurons σ_{c_7} and σ_{c_8} each have a spike, they can not be applied. At time $t_1 + 3$, with two spikes in neuron σ_{c_9}, it applies rule $2|a^x \to a^{\beta(x)}$ to send two spikes to neuron σ_1. The process is repeated until the second spike arrives at neurons σ_{c_5} and σ_{c_6}. During each computation step, two spikes in neuron σ_{c_9} are constantly sent to neuron σ_1. Therefore, from time t_1+3 to time $t_1+g(x)+2$, neuron σ_1 receives in total $2g(x)$ spikes (i.e., the number of spikes in register 1 is $g(x)$).

Suppose that neuron σ_{in} receives the second spike at time t_2 (in fact, $t_2 = t_1+g(x)+2$). Therefore, neuron σ_{in} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons $\sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}$ and σ_{c_4} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t_2 + 1$, neuron σ_{c_2} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_5} and σ_{c_6} (because of $x = 1$ and $\beta(x) = 1$). At the same time, neuron σ_{c_4} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_7} and σ_{c_8} (because of $x = 1$ and $\beta(x) = 1$). Thus, neurons σ_{c_7} and σ_{c_8} each have two spikes. However, neurons σ_{c_1} and σ_{c_3} can not be applied. At time $t_2 + 2$, with two spikes in neurons σ_{c_5} and σ_{c_6}, they each apply rule $2|a^x \to \lambda$ to remove the two spikes. At the same time, neurons σ_{c_7} and σ_{c_8} each apply rule $1|a^x \to a^{\beta(x)}$ to send two spikes to each other, and they send two spikes to neuron σ_{c_9} respectively (because of $x = 2$ and $\beta(x) = 2$). Thus, neuron $\sigma_{c_{10}}$ receives four spikes. At time t_2+3, with four spikes in neuron $\sigma_{c_{10}}$, it applies rule $4|a^x \to a^{\beta(x)/2}$ to send two spikes to neuron σ_2 (because of $x = 4$ and $\beta(x)/2 = 2$). The process is repeated until the third spike arrives at neurons σ_{c_7} and σ_{c_8}. During each computation step, two spikes in neuron $\sigma_{c_{10}}$ are constantly sent to neuron σ_2. Therefore, from time $t_2 + 3$ to time $t_2 + y + 2$, neuron σ_2 receives in total $2y$ spikes (i.e., the number of spikes in register 1 is y).

After neuron σ_{in} receives the third spike, with three spikes in neuron σ_{c_1}, it applies rule $3|a^x \to a^{\gamma(x)}$ to send two spikes to neuron σ_{l_0} (because of $x = 3$ and $\gamma(x) = 2$), meaning that the system starts the simulation of initial instruction l_0. At the same time, neuron σ_{c_2} applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_5} and σ_{c_6}, and with three spikes in neuron σ_{c_3}, it applies rule $3|a^x \to a^{\gamma(x)-1}$ to send a spike to neurons σ_{c_5} and σ_{c_6}. Thus, neurons σ_{c_5} and σ_{c_6} each receive two spikes, which will be removed by rule $2|a^x \to \lambda$ at next time. At the same time, with three spikes in neurons σ_{c_7} and σ_{c_8}, they apply rule $3|a^x \to \lambda$ to remove the three spikes.

From Fig. 1, we can observe that all ADD instructions have the form $l_i : (ADD(r), l_j)$. Consequently, a deterministic ADD module can be used in the simulation of ADD instruction, shown in Fig. 7. The working principle of the deterministic ADD module has been discussed in the proof of Theorem 4.2.

The SUB module in Fig. 4 is used in the simulation of SUB instruction $l_i : (SUB(r), l_j, l_k)$. The working principle of the SUB module has been described in the proof of Theorem 4.1.

Suppose that M'_u halts now, that is, the instruction l_h is reached. The computation result is contained in register 8, and during the computation it is never decreased. The computation result is exported by an OUTPUT module, shown in Fig. 10.

Suppose that neuron σ_{l_h} receives two spikes at time t, meaning that the computation in M'_u halts (the halting instruction l_h is reached), and neuron σ_8 contains $2n$ spikes (indicating that register 8 of M'_u has the number n). With two spikes in neuron σ_{l_h}, rule $2|a^x \to a^{\beta(x)}$ is applied to send two spikes to neuron σ_{c_1} (because of $x = 2$ and $\beta(x) = 2$). At time $t + 1$, with two spikes in neuron σ_{c_1}, rule $2|a^x \to a^{\beta(x)/2}$ is applied to send a spike to neurons σ_8 and σ_{out} respectively (because of $x = 2$ and $\beta(x)/2 = 1$).

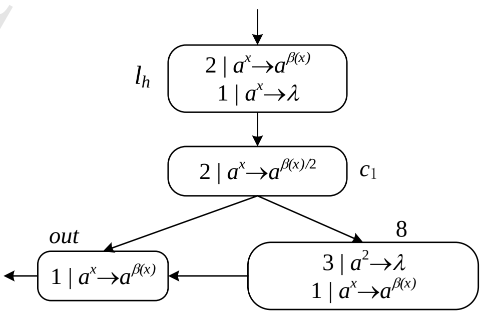

Fig. 10. OUTPUT module.

At time $t + 2$, with a spike in neuron σ_{out}, rule $1|a^x \to a^{\beta(x)}$ is applied to send the first spike to the environment (because of $x = 1$ and $\beta(x) = 1$). Since neuron σ_8 receives a spike, the value of spikes in it becomes odd, hence, rule $3|a^2 \to \lambda$ or rule $1|a^x \to a^{\beta(x)}$ can be applied. If neuron σ_8 contains $(2n + 1) \geq 3$ spikes, then rule $3|a^2 \to \lambda$ is applied based on maximum threshold strategy. Thus, at each step from time $t + 2$ to time $t + n$, neuron σ_8 applies rule $3|a^2 \to \lambda$ to remove two spikes.

At time $t + n + 1$, neuron σ_8 contains only one spike, hence, rule $1|a^x \to a^{\beta(x)}$ is applied to send a spike to neuron σ_{out} (because of $x = 1$ and $\beta(x) = 1$). At $t + n + 2$, with a spike in neuron

σ_{out}, rule $1|a^x \to a^{\beta(x)}$ is applied to send the second spike to the environment (because of $x = 1$ and $\beta(x) = 1$). Therefore, the time interval between the two spikes sent to the environment by neuron σ_{out} is $(t+n+2) - (t+2) = n$, which is exactly the number stored in register 8 when the computation of M'_u halts.

From the discussion above, system Π_3 correctly simulates register machine M'_u. In NSNP system Π_3, we use a total of 117 neurons: (i) 11 neurons for the INPUT module, (ii) 2 neuron for the OUTPUT module, (iii) 70 auxiliary neurons for 14 SUB instructions, (iv) 9 neurons for 9 registers, and (v) 25 neurons for 25 instructions. □

5.2. *Small universal NSNP systems as number generator*

Based on Theorem 4.1 and Theorem 5.1, we will construct a small universal NSNP system as number generator. The small universal NSNP system as number generator will be proven by simulating small universal register machine M'_u. In the case of number generator, an NSNP system corresponds to a set of numbers, $N_2(\Pi)$. In this way, given a fixed admissible enumeration of the unary partial recursive functions, $(\varphi_0, \varphi_1, \ldots)$, there is a recursive function g such that for each natural number n, the set of numbers generated by the NSNP system is equal to $\{n \in N|\varphi_x(n) \text{ is defined}\}$ after the number $g(x)$ is introduced into the system (by reading the sequence $10^{g(x)}1$ from the environment)[51]. At this time, the NSNP system is said to be small universal. In addition, after the "code" $g(x)$ of the partial recursive function φ_x is introduced into a specified neuron, the system generates all numbers n for which $\varphi_x(n)$ is defined.

Theorem 5.2. *There exists a small universal NSNP system having 164 neurons as a number generator.*

Proof. We construct an NSNP system Π_4 as number generator for simulating the small universal register machine M'_u. Different from NSNP system for computing function above, the following three aspects are considered in NSNP system Π_4 as number generator:

(1) Read the spike train $10^{g(x)-1}1$ from the environment and load $2g(x)$ spikes into neuron σ_1 (indicating that register 1 has the number $g(x)$);
(2) Load non-deterministically arbitrary natural number n into neurons σ_2 and σ_8;
(3) Check whether the function φ_x is defined for n. For this purpose, start register machine M'_u, with $g(x)$ spikes in register 1 and n in registers 2 and 8. If the computation of $g(x)$ halts, then n is introduced in the set of generated numbers.

Fig. 11 gives the INPUT module. The module reads a spike train $10^{g(x)-1}1$ from the environment, where $2g(x)$ spikes are stored in neuron σ_1, while $2n$ spikes are placed in neurons σ_2 and σ_8 respectively.

Suppose that at time t, neuron σ_{in} receives the first spike from the environment. With a spike in neuron σ_{in}, rule $1|a^x \to a^{\beta(x)}$ is applied to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3} respectively (because of $x = 1$ and $\beta(x) = 1$). At time $t+1$, neurons σ_{c_2} and σ_{c_3} each apply rule $1|a^x \to a^{\beta(x)}$ to send a spike to neuron σ_1 (because of $x = 1$ and $\beta(x) = 1$). Hence, neuron σ_1 receives two spikes. Since that neuron σ_{c_1} has only one spike, rule $2|a^x \to a^{\beta(x)}$ can not be applied.

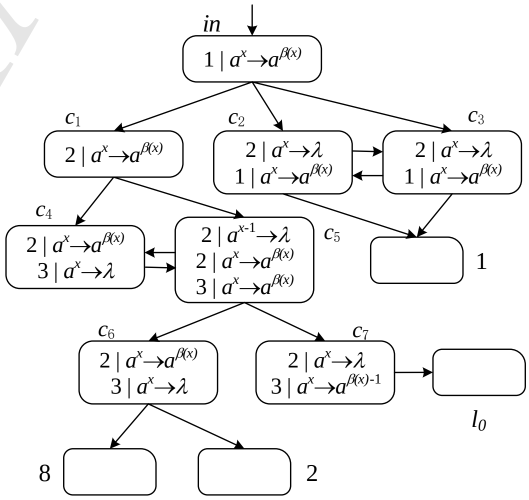

Fig. 11. INPUT module.

From time $t+1$ until neuron σ_{in} receives the second spike, neurons σ_{c_2} and σ_{c_3} each send a spike to neuron σ_1. Therefore, from time $t+1$ to time $t + g(x) + 1$, neuron σ_1 receives $2n$ spikes in total (meaning that the number stored in register 1 is $g(x)$).

At time $t+g(x)+1$, neuron σ_{in} receives the second spike, and then it fires and applies rule $1|a^x \to a^{\beta(x)}$ to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3} (because of $x = 1$ and $\beta(x) = 1$). At time $t + g(x) + 2$, with two spikes in neuron σ_{c_1}, rule $2|a^x \rightarrow a^{\beta(x)}$ is applied to send two spikes to neurons σ_{c_4} and σ_{c_5} respectively (because of $x = 2$ and $\beta(x) = 2$). At time $t + g(x) + 3$, with two spikes in neuron σ_{c_4}, rule $2|a^x \rightarrow a^{\beta(x)}$ is applied to send two spikes to neuron σ_{c_5} (because of $x = 2$ and $\beta(x) = 2$). Since neuron σ_{c_5} has two spikes, rule $2|a^x \rightarrow a^{\beta(x)}$ and rule $2|a^{x-1} \rightarrow \lambda$ can be applied simultaneously. Hence, one of them is non-deterministically chosen and applied. If rule $2|a^x \rightarrow a^{\beta(x)}$ in neuron σ_{c_5} is applied, it sends two spikes to neurons σ_{c_4}, σ_{c_6} and σ_{c_7}; otherwise, if rule $2|a^{x-1} \rightarrow \lambda$ in neuron σ_{c_5} is applied, the module will complete the input operation. Therefore, suppose that, from time $t+g(x)+3$ to time $t + g(x) + n + 3$, rule $2|a^x \rightarrow a^{\beta(x)}$ in neuron σ_{c_5} is applied constantly. At each step during this period, neurons σ_{c_4} and σ_{c_5} send two spikes to each other, and neuron σ_{c_5} also sends two spikes to neurons σ_{c_6} and σ_{c_7}. And then, with two spikes in neuron σ_{c_6}, rule $2|a^x \rightarrow a^{\beta(x)}$ is applied to send two spikes to neurons σ_1 and σ_8 (because of $x = 2$ and $\beta(x) = 2$). However, two spikes in neuron σ_{c_7} are removed by rule $2|a^x \rightarrow \lambda$. Therefore, during this period, neurons σ_1 and σ_8 each receive $2n$ spikes.

Suppose that at time $t + g(x) + n + 4$, rule $2|a^{x-1} \rightarrow \lambda$ in neuron σ_{c_5} is applied to remove a spike (retaining a spike). At the same time, neuron σ_{c_6} applies rule $2|a^x \rightarrow a^{\beta(x)}$ to send two spikes to neuron σ_{c_7}. At time $t + g(x) + n + 5$, with three spikes in neuron σ_{c_5}, rule $3|a^x \rightarrow a^{\beta(x)}$ is applied to send three spikes to neurons σ_{c_4}, σ_{c_6} and σ_{c_7} (because of $x = 3$ and $\beta(x) = 3$). At time $t + g(x) + n + 6$, with three spikes in neurons σ_{c_4} and σ_{c_6}, the three spikes are removed by rule $3|a^x \rightarrow \lambda$. At the same time, neuron σ_{c_7} applies rule $3|a^x \rightarrow a^{\beta(x-1)}$ to send two spikes to neuron σ_{l_0} (because of $x = 2$ and $\beta(x-1) = 2$). With two spikes in neuron σ_{l_0}, the system starts the simulation of initial instruction l_0.

NSNP system Π_4 uses ADD module in Fig. 3, SUB module in Fig. 4 and OUTPUT module in Fig. 11. These modules have be described in Theorem 4.1, Theorem 4.2 and Theorem 5.1, respectively.

Consequently, NSNP system Π_4 contains a total of 164 neurons: (i) 8 neurons for the INPUT module, (ii) 2 neuron for the OUTPUT module, (iii) 50 auxiliary neurons for 10 ADD instructions, (iv) 70 auxiliary neurons for 14 SUB instructions, (v) 9 neurons for 9 registers, and (vi) 25 neurons for 25 instructions. □

5.3. *Discussion*

Theorem 5.1 shows a small number of computing units (i.e., neurons) for NSNP systems as function-computing devices to achieve Turing universality. To further evaluate the computational power of NSNP systems, Table 1 compares the proposed variant with other computing models in terms of small numbers of computing units. From Table 1, we can observe that recurrent neural networks [61], PSNP systems [34], and SNQ P systems with one type of spike [36] need 886, 200, and 181 neurons, respectively, to achieve Turing universality for the computing function, and NSNP systems need fewer neurons than all of these.

Table 1. Comparison of different computing models in terms of small numbers of computing units.

Computing models	Number of neurons
NSNP systems	117
PSNP systems [34]	200
SNQ P systems [36]	181
Recurrent neural networks [61]	886

6. Conclusions and Further Work

A new variant of SNP systems, NSNP systems, was discussed in this work. In addition to using a real state, NSNP systems significantly differ from usual SNP systems due to nonlinear spiking rules: the consumed and generated amount of spikes are often the nonlinear function of the states of neurons. This paper investigated the computational power of NSNP systems. The universalities of NSNP systems as number generating device and number accepting device have been proven. Moreover, we established two small universality results of NSNP systems for computing function and number generating: a small universal function computing device consisting of 117 neurons and a small universal number generator consisting of 164 neurons were constructed.

As stated in the existing SNP systems, some problems that refer to NSNP systems will be investigated, for example, language generator, and sequential and asynchronous modes. Moreover, it is worth studying how to combine other mechanisms in NSNP systems to propose more models. In addition, NSNP systems can provide a nonlinear processing ability.

Therefore, our a future work will focus on another topic of NSNP systems, i.e., application of NSNP systems in some real-world problems.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (No. 61472328), Research Fund of Sichuan Science and Technology Project (No. 2018JY0083), Research Foundation of the Education Department of Sichuan province (No. 17TD0034), and Innovation Fund of Postgraduate, Xihua University (Nos. YCJJ2019019 and YCJJ2019020), China.

References

1. Gh. Păun, Computing with membranes, *Journal of Computer System Sciences* 61(1) (2000) 108-143.
2. Gh. Păun, G. Rozenberg and A. Salomaa, *The Oxford Handbook of Membrance Computing* (New York: Oxford University Press, 2010).
3. H. Peng, J. Wang, P. Shi, M.J. Pérez-Jiménez and A. Riscos-Núñez, An extended membrane system with active membrane to solve automatic fuzzy clustering problems, *International Journal of Neural Systems* 26(3) (2016) 1650004: 1-17.
4. H. Peng, P. Shi, J. Wang, A. Riscos-Núñez and M.J. Pérez-Jiménez, Multiobjective fuzzy clustering approach based on tissue-like membrane systems, *Knowledge-Based Systems* 125 (2017) 74-82.
5. M. Ionescu, Gh. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* 71 (2006) 279-308.
6. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *International Journal of Neural Systems* 19(4) (2009) 295-308.
7. X. Zhang, G. Foderaro, C. Henriquez, and S. Ferrari, A Scalable weight-free learning algorithm for regulatory control of cell activity in spiking neuronal networks, *International Journal of Neural Systems* 28(2) (2018) 1750015 (20 pages).
8. G. Antunes, S.F. Faria Da Silva, and F.M. Simoes De Souza, Mirror neurons modeled through spike-timing dependent plasticity are affected by channelopathies associated with autism spectrum disorder, *International Journal of Neural Systems* 28(5) (2018) 1750058 (15 pages).
9. A. Antonietti, J. Monaco, E. D'Angelo, A. Pedrocchi, and C. Casellato, Dynamic redistribution of plasticity in a cerebellar spiking neural network reproducing an associative learning task perturbed by TMS, *International Journal of Neural Systems* 28(9) (2018) (19 pages).
10. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Networks* 22(10) (2009) 1419-1431.
11. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for eeg classification and epilepsy and seizure detection, *Integrated Computer-Aided Engineering* 14(3) (2007) 187-212.
12. M. Bernert and B. Yvert, An attention-based spiking neural network for unsupervised spike-sorting, *International Journal of Neural Systems* 29(8) (2019) 1850059 (18 pages).
13. F. Galan-Prado, J. Font, and J.L. Rossello, Compact hardware synthesis of stochastic spiking neural networks, *International Journal of Neural Systems* 29(8) (2019) 1950004 (13 pages).
14. R. Hu, S. Chang, Q. Huang, H. Wang, and J. He, Monitor-based spiking recurrent network for the representation of complex dynamic patterns, *International Journal of Neural Systems* 29(8) (2019) 1950006 (22 pages).
15. Y. Todo, Z. Tang, H. Todo, J. Ji, and K. Yamashita, Neurons with multiplicative interactions of nonlinear synapses, *International Journal of Neural Systems* 29(8) (2019) 1950012 (18 pages).
16. A. Geminiani, C. Casellato, A. Antonietti, E. D'Angelo, and A. Pedrocchi, A multiple-plasticity spiking neural network embedded in a closed-loop control system to model cerebellar pathologies, *International Journal of Neural Systems* 28(5) (2018) 1750017 (23 pages).
17. B. Song, L. Pan and M.J. Pérez-Jiménez, Cell-like P systems with channel states and symport/antiport rules, *IEEE Transactions on Nanobioscience* 15(6) (2016) 555-566.
18. X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate and R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, *Integrated Computer-Aided Engineering* 23(1) (2016) 15-30.
19. R. Freund, Gh. Păun and M.J. Pérez-Jiménez, Tissue-like P systems with channel-states, *Theoretical Computer Science* 330(1) (2005) 101-116.
20. F. Bernardini and M. Gheorghe, Cell communication in tissue P systems: universality results, *Soft Computing* 9(9) (2005) 640-649.
21. H. Peng, J. Wang, M.J. Pérez-Jiménez and A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, *Information Sciences* 304 (2015) 80-91.
22. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca and C. Zandron, Tissue P systems with small cell volume, *Fundamenta Informaticae* 154(1-4) (2017) 261-275.
23. F. Bernardini, M. Gheorghe, Population P systems, *Journal of Universal Science* 10(5) (2004) 509-539.
24. E. Sánchez-Karhunen, L. Valencia-Cabrera, Modelling complex market interactions using PDP systems, Journal of Membrane Computing 1(1) (2019)

40-51.
25. L. Cienciala, L. Ciencialová and E. Csuhaj-Varjú, A class of restricted P colonies with string environment, *Natural Computing* 15(4) (2016) 1-9.
26. Gh. Păun, Spiking neural P systems with astrocyte-like control, *Journal of Universal Computer Science* 13(11) (2007) 1707-1721.
27. L. Pan, J. Wang and H.J. Hoogeboom, Spiking neural P systems with astrocytes, *Neural Computation* 24(3) (2012) 805-825.
28. L. Pan and Gh. Păun, Spiking neural P systems with anti-spikes, *Int. J. of Computers, Communications & Control* IV(3) (2009) 273-282.
29. H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Luo and X. Huang, Spiking neural P systems with multiple channels, *Neural Networks* 95 (2017) 66-71.
30. X. Song, J. Wang, H. Peng, G. Ning, Z. Sun, T. Wang and F. Yang, Spiking neural P systems with multiple channels and anti-spikes, *Biosystems* . 169C170 (2018) 13-19.
31. T. Song, L. Pan and Gh. Păun, Spiking neural P systems with rules on synapses, *Theoretical Computer Science* 529 (2014) 82-95.
32. H. Peng, R. Chen, J. Wang, X. Song, T. Wang, F. Yang, Z. Sun, Competitive spiking neural P systems with rules on synapses, *IEEE Transactions on NanoBioscience* 16(8) (2017) 888-895.
33. H.M. Chen, T.-O. Ishdorj, Gh. Păun, Computing along the axon, *Progress in Natual Science* 17(4) (2007) 417-423.
34. T. Wu, A. Păun, Z. Zhang, L. Pan, Spiking neural P systems with polarizations, IEEE Transactions on Neural Networks and Learning Systems 29(8) (2017), 3349-3360.
35. F.G.C. Cabarle, H.N. Adorna, M. Jiang, X. Zeng, Spiking neural p systems with scheduled synapses, IEEE Transactions on Nanobioscience 27(5) (2016) 1337-1347.
36. L. Pan, Gh. Păun, G. Zhang, F. Neri, Spiking neural P systems with communication on request, International Journal of Neural Systems 27(8) (2017), 1750042: 1-13.
37. O.H. Ibarra, A. Păun and A. Rodríguez-Paťon, Sequential SNP systems based on min/max spike number, *Theoretical Computer Science* 410(30) (2009) 2982-2991.
38. X. Zhang, X. Zeng, B. Luo and L. Pan, On some classes of sequential spiking neural P systems, *Neural Computation* 26(5) (2014) 974-997.
39. J. Wang, H.J. Hoogeboom, L. Pan, Gh. Păun and M.J. Pérez-Jiménez, Spiking neural P systems with weights, *Neural Computation* 22 (2010) 2615-2646.
40. H. Peng, J. Wang, M.J. Pérenz-Jiménez, A. Riscos-Núñez, Dynamic threshold neural P systems, Knowledge-Based Systems 163 (2019) 875-884.
41. H. Peng, J. Wang, Coupled neural P systems, IEEE Transactions on Neural Networks and Learning Systems 30(6) (2019) 1672-1682.
42. X. Zeng, X. Zhang, T. Song and L. Pan, Spiking neural P systems with thresholds, *Neural Computation* 26(7) (2014) 1340-1361.
43. M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Egecioglu, M. Ionescu and S. Woodworth, Asynchronous spiking neural P systems, *Theoretical Computer Science* 410(24) (2009) 2352-2364.
44. T. Song, L. Pan and Gh. Păun, Asynchronous spiking neural P systems with local synchronization, *Information Sciences* 219 (2012) 197-207.
45. H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao and T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, *Information Sciences* 235(20) (2013) 106-116.
46. J. Wang, P. Shi, H. Peng, M.J. Pérez-Jiménez and T. Wang, Weighted fuzzy spiking neural P system, *IEEE Trans. Fuzzy Syst.* 21(2) (2013) 209-220.
47. J. Wang, H. Peng, W. Yu, J. Ming, M.J. Pérenz-Jiménez, C. Tao, X. Huang, Interval-valued fuzzy spiking neural P systems for fault diagnosis of power transmission networks, Engineering Applications of Artificial Intelligence 82 (2019) 102C109.
48. T. Song, J. Xu and L. Pan, On the universality and non-universality of spiking neural p systems with rules on synapses, *IEEE Transactions on Nanobioscience* 14(8) (2015) 960-966.
49. T. Wu, F.D. Bîlbîe, A. Păun, L. Pan, F. Neri. Simplified and yet turing universal spiking neural p systems with communication on request, International Journal of Neural Systems 28(8) (2018), 1850013: 1-19.
50. H. Chen, R. Freund, M. Ionescu, Gh. Păun and M.J. Pérez-Jiménez, On string languages generated by spiking neural P systems, *Fundamenta Informaticae* 75(1) (2007) 141-162.
51. A. Păun and Gh. Păun, Small universal spiking neural P systems, *BioSystems* 90(1) (2007) 48-60.
52. L. Pan, Gh. Păun and M.J. Pérez-Jiménez, "Spiking neural P systems with neuron division and budding," *Science China Information Sciences* 54(8) (2011) 1596-1607.
53. A. Leporati, G. Mauri, C. Zandron, Gh. Păun and M.J. Pérez-Jiménez, Uniform solutions to SAT and Subset Sum by spiking neural P systems, *Natural Computing* 8(4) (2009) 681-702.
54. T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, M.J. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, IEEE Transactions on Power Systems 30(3) (2015) 1182-1194.
55. H. Peng, J. Wang, P. Shi, M.J. Prez-Jiménez, A. Riscos-Núñez, Fault diagnosis of power systems using fuzzy tissue-like P systems, Integrated Computer-Aided Engineering 24(4) (2017) 401-411.
56. H. Peng, J. Wang, J. Ming, P. Shi, M.J. Pérez-Jiménez, W. Yu, C. Tao, Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems, IEEE Transactions on Smart Grid 9(5) (2018)

4777-4784.

57. D. Díaz-Pernil, F. Peña-Cantillana and M.A. Gutiérrez-Naranjo, A parallel algorithm for skeletonizing images by using spiking neural P systems, *Neurocomputing* 115 (2013) 81-91.
58. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, H. Peng, Membrane computing and image processing: a short survey, Journal of Membrane Computing 1(1) (2019) 58-73.
59. G. Zhang, H. Rong, F. Neri and M.J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems* 24(5) (2014) 1440006: 1-16.
60. I. Korec, Small universal register machines, *Theoretical Computer Science* 168(2) (1996) 267-301.
61. H.T. Siegelmann and E.D. Sontag, On the computational power of neural nets, Journal of Computer and System Sciences 50(1) (1995) 132-150.