Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Spiking neural P systems with structural plasticity and anti-spikes ☆

Qian Yang [a], Bo Li [a], Yue Huang [a], Hong Peng [a,*], Jun Wang [b]

[a] School of Computer and Software Engineering, Xihua University, Chengdu, Sichuan, 610039, China
[b] School of Electrical Engineering and Electronic Information, Xihua University, Chengdu, Sichuan, 610039, China

ARTICLE INFO

Article history:
Received 19 May 2019
Received in revised form 1 August 2019
Accepted 28 August 2019
Available online xxxx
Communicated by M.J. Pérez-Jiménez

Keywords:
Membrane computing
Spiking neural P systems
Turing universality
Structural plasticity
Anti-spike

ABSTRACT

Spiking neural P systems (in short, SNP systems) are a class of distributed parallel computing devices, abstracted from the way neurons communicate by means of spikes. This paper discusses spiking neural P systems with structural plasticity and anti-spikes (in short, SNP-SPA systems), a new variant of SNP systems with two interesting features: structural plasticity and anti-spike. By means of plasticity rules in neurons, SNP-SPA systems can provide a dynamic directed graph structure. Turing universality of SNP-SPA systems is discussed. It is proven that SNP-SPA systems as number generating/accepting devices are Turing universal, and a small example with 56 neurons that computes a universal function is constructed. The introduction of anti-spikes allows to reduce the modules in the proof of universality.

1. Introduction

Membrane computing is a class of distributed and parallel computing models inspired by biological cells [1,2], known as P systems or membrane systems. From a topological view, P systems can be of one out of three main types: cell-like P system (its structure is hierarchical), tissue-like P system (its structure is network-like), and neural-like P system (its structure is a directed graph). P systems have been widely investigated [3–5], and most of them can be used as universal number generating/accepting devices as well as function computing devices. Some P systems can be used as universal language-generating devices and could be mentioned that there is a space-time trade-off to develop efficient solutions to computationally hard problems [6–10]. In addition, P systems have been successfully used to solve some real-world problems [11–16].

Spiking neural P systems (in short, SNP systems) are a class of neural-like P systems, abstracted from the mechanism of spiking neurons [17]. An SNP system has a topological structure of directed graph, where the neurons are the nodes of the directed graph and the arcs represent its synapses. Abstracting from different mechanisms of biological nervous systems and/or using the ideas in computer sciences, many variants of SNP systems were investigated, such as SNP systems with astrocytes [18,19], SNP systems with anti-spikes [20,21], SNP systems with weights [22], SNP systems with thresholds [23], SNP systems with rules on synapses [24–26], SNP systems with multiple channels [27,28], SNP systems with

☆ This work was partially supported by the Research Fund of Sichuan Science and Technology Project (No. 2018JY0083), Chunhui Project Foundation of the Education Department of China (Nos. Z2016143 and Z2016148), Research Foundation of the Education Department of Sichuan Province (No. 17TD0034), and the Innovation Fund of Postgraduate, Xihua University (No. YCJJ2018090), China.

* Corresponding author.
E-mail address: ph.xhu@hotmail.com (H. Peng).

https://doi.org/10.1016/j.tcs.2019.08.034
0304-3975/

polarizations [29], coupled neural P systems [30], dynamic threshold neural P systems [31], SNP systems with scheduled synapses [32], SNP systems with colored spikes [33], and so on. In addition, several working modes were discussed, including asynchronous mode [34], asynchronous mode with local synchronization [35], sequential mode [36]. It was proven that most variants of SNP systems have computational completeness (equivalent to Turing machine) [37,38]. In addition, fuzzy logic was introduced into SNP system to propose several fuzzy spiking neural P systems (in short, FSNP systems) [39–41], and they have been applied to fault diagnosis [42–44].

As usual, SNP systems have rules of two types: spiking and forgetting rules. These rules are used to evolve the objects in the systems. From an application perspective, SNP systems are a kind of dynamic systems. However, their structures (directed graph) are static, that is, their topology is not changing during the computation. Recently, dynamic topological structure has received attention, for example, spiking neural P systems with structural plasticity (in short, SNPSP systems) [45–48]. In SNPSP systems, plasticity rules of the form $E/a^c \to \alpha k(i, N)$ are introduced to create or delete the synapses. Therefore, SNPSP systems have a dynamic topological structure.

From the perspective of the objects, regular SNP systems use a type of spikes (denoted by a), and the status of a neuron is characterized by the number of spikes. Compared with other types of P systems (such as cell-like P systems and tissue-like P systems), SNP systems have weaker expression ability, although most of them have been proven to be Turing universal. Thus, a few works have been devoted to extend SNP systems, such as spiking neural P systems with anti-spikes (ASNP systems) [20] and variants [21,28]. In ASNP systems, another type of spike ($\bar{a}$) is introduced, called anti-spike. It was indicated that the introduction of anti-spikes can reduce the modules in the proofs of universality.

This paper focuses on the extension of SNPSP systems. By introducing the anti-spikes, a new variant is investigated, called spiking neural P system with structural plasticity and anti-spikes (in short, SNP-SPA systems). SNP-SPA systems have two interesting features: structural plasticity and a pair of spikes and anti-spikes. Except the use of spiking rules over $\{a, \bar{a}\}$, regular plasticity rules are extended to the form of $E/b^c \to b'; \alpha k(i, N)$, where $b, b' \in \{a, \bar{a}\}$. Therefore, SNP-SPA systems are a kind of distributed parallel computation models with dynamic structure. Moreover, due to the use of a pair of spikes and anti-spikes, SNP-SPA systems can provide better expression ability. This is the motivation of proposing the variant. However, we focus on the computational completeness of SNP-SPA systems in this paper, including Turing universality as number generating/accepting devices and function computing devices.

The remainder of this paper is arranged as follows. In Section 2, we review some mathematical preliminaries that will be used in the following. Section 3 gives the definition of SNP-SPA systems and provides an illustration example. Section 4 discusses the computational power of SNP-SPA systems as number generating/accepting devices and function computing devices. Finally, a conclusion is drawn in Section 5.

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

A regular expression over V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. 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 $a \in V$, (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 V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

A non-deterministic register machine can be described by a tuple $M = (m, H, l_0, l_h, I)$. In the register machine M, m is the number of registers, H denotes the set of instruction labels, l_0 is the starting label, l_h is the halting label, 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 to instruction l_j or instruction l_k non-deterministically);
(2) $l_i : (SUB(r), l_j, l_k)$ (if the number contained in register r is greater than 0, decrease 1 from it and then go to instruction l_j; otherwise, go directly to instruction l_k);
(3) $l_h : HALT$ (halting instruction).

Let us denote by $N_{gen}(M)$ the set of numbers that are non-deterministically generated by register machine M. Its working mechanism can be illustrated 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. M starts from instruction l_0 and then continues to execute other instructions. When it turns to instruction l_h, the computation is accomplished, and the final result is saved in the first register.

The register machine M can be also used to accept numbers. Let us denote by $N_{acc}(M)$ the set of numbers accepted by M. Its working mechanism can be illustrated as follows. At the beginning, all registers are empty except the first register,

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

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.

and a number is introduced into the first register. In accepting mode, register machine is deterministic, meaning that $l_i : (ADD(r), l_j)$ is used to substitute $l_i : (ADD(r), l_j, l_k)$ as the ADD instruction.

It is well-known that a register machine M can generate/accept any Turing computable number set (denoted by NRE).

The register machine $M = (m, H, l_0, l_h, I)$ can compute functions $f : N^k \to N$. It works as follows: initially, all registers are empty and k parameters are introduced into k specific registers; M begins with instruction l_0 and ends with instruction l_h respectively; when the register machine halts, the computed function value is stored in register r_t. In computing mode, the register machine is deterministic.

A small universal register machine $M_u = (8, H, l_0, l_h, I)$ for computing functions is introduced in [49]. The register machine contains 8 registers and 23 instructions. By introducing numbers $g(x)$ and y into registers 1 and 2 respectively, the register machine is able to compute any function $\varphi_x(y)$. The computed function value is placed in register 0 when register machine M_u stops. Some modification to the register machine is made for the convenience of construction: a new register with label 8 is introduced, and the halting instruction is substituted by $l_{22} : (SUB(0), l_{23}, l_{24})$, $l_{23} : (ADD(8), l_{22})$ and $l'_h : HALT$. The computed value is copied from register 0 to register 8 after register machine halts, and no SUB instruction acts on the register that stores the computation result. The modified register machine is denoted by $M'_u = (9, H, l_0, l_h, I)$, shown in Fig. 1.

3. SNP-SPA systems

In this section, the definition of spiking neural P system with structural plasticity and anti-spikes (SNP-SPA systems, in short) is firstly given, and then an example is provided to illustrate its working mechanism.

3.1. Definition

Definition 1. An SNP-SPA system, of degree $m \geq 1$, is defined as a tuple

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

where

(1) $O = \{a, \bar{a}\}$ is an alphabet (a and $\bar{a}$ denote the spike and anti-spike, respectively);
(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ 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 the neuron σ_i;
  (b) R_i is the finite set of rules with the following two forms:
    (i) Spiking rules: $E/b^c \to b'$, where E is a regular expression over O, $b, b' \in \{a, \bar{a}\}$, $c \geq 1$;
    (ii) Plasticity rule: $E/b^c \to b'; \alpha k(i, N_j)$, where $b, b' \in \{a, \bar{a}\}$, $\alpha \in \{+, -, \pm, \mp\}$, $c \geq 1$, $k \geq 1$, $1 \leq j \leq |R_j|$, and $N_j \subseteq \{1, 2, \ldots, m\}$;
(4) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $\forall 1 \leq i \leq m$, are synapses between neurons;
(5) $in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neurons, respectively.

In an SNP-SPA system, each neuron contains one or more spikes or anti-spikes, however, the two types of spikes do not occur in the same neuron at the same time. For neuron σ_i, denote by $Pre(i) = \{j | (j, i) \in syn\}$ the set of its precursor neurons and by $Suc(i) = \{j | (i, j) \in syn\}$ the set of its successor neurons. The SNP-SPA system has two types of rules: (i) spiking rule $E/b^c \to b'$; (ii) plasticity rule $E/b^c \to b'; \alpha k(i, N_j)$. The semantics of the spiking rule can be described as follows. If neuron σ_i contains n spikes, $b^n \in L(E)$ and $n \geq c$, then rule $E/b^c \to b'$ can be applied, where $b, b' \in \{a, \bar{a}\}$. When the spiking rule is applied, c spikes or anti-spikes are consumed and then a spike or an anti-spike is generated. The generated spike or anti-spike is sent to its successor neurons, i.e., neurons in $Suc(i)$. Note that there is an implicit rule $\bar{a}a \to \lambda$ in SNP-SPA system, and as usual, it is assumed that the implicit rule is executed before spiking rules and plasticity rules.

Suppose that the neuron σ_i has a plasticity rule $E/b^c \to b'; \alpha k(i, N_j)$. The semantics of plasticity rule can be illustrated as follows. If neuron σ_i contains n spikes or anti-spikes, $b^n \in L(E)$ and $n \geq c$, then rule $E/b^c \to b'; \alpha k(i, N_j)$ can be applied,

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

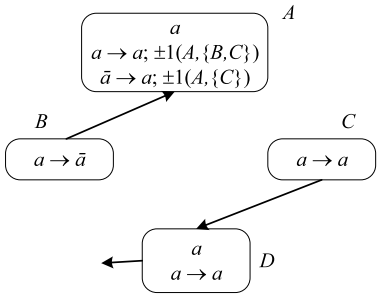

Fig. 2. An SNP-SPA system, where neuron σ_D is the output neuron.

where $b, b' \in \{a, \bar{a}\}$, and N_j is the set of neurons that can connect to or disconnect from neuron σ_i by the plasticity rule. As in [18], when the plasticity rule is applied, c spikes or anti-spikes are consumed and a spike or an anti-spike is generated, and then α is used to decide whether synapses are created or deleted and how the generated spike or anti-spike is transmitted.

(1) If $\alpha = +$ and $N_j - Suc(i) = \emptyset$, or if $\alpha = -$ and $Suc(i) = \emptyset$, then the plasticity rule is applied to consume c spikes or anti-spikes, but no synapse is created or deleted.
(2) If $\alpha = +$ and $|N_j - Suc(i)| \leq k$, then for each $\sigma_l \in N_j - Suc(i)$, a new synapse between σ_i and σ_l is created. If $|N_j - Suc(i)| > k$, then k neurons are chosen nondeterministically from $N_j - Suc(i)$, and for each chosen neuron σ_l, a new synapse between σ_i and σ_l is created.
(3) If $\alpha = -$ and $|Suc(i)| \leq k$, then all synapses $\{(i, l) \mid \sigma_l \in Suc(i)\}$ are deleted. If $|Suc(i)| > k$, then k synapses are chosen nondeterministically from $\{(i, l) \mid \sigma_l \in Suc(i)\}$, and then they are deleted.
(4) If $\alpha = \pm$, the synapses are created at time t, and at time $t + 1$ they are deleted. If $\alpha = \mp$, the synapses are deleted at time t, and at time $t + 1$ they are created again.

As in SNP systems, all neurons work in parallel under the control of a global clock, however, the rules in each neuron are used sequentially. In each computation step, if a neuron can use a rule, then the rule must be applied. If several rules can be applied in a neuron, then one of them is chosen nondeterministically and applied. It is worth noting that when a plasticity rule is applied in a neuron, one or more synapses are created and a spike or an anti-spike is sent to its all successor neurons, or they are deleted.

As in [47], SNP-SPA system has a directed graph with dynamic structure because of plasticity rules. The configuration of the system is described by the number of spikes or anti-spikes present in each neuron. The initial configuration is denoted by $(n_1, n_2, \ldots, n_m)$. Applying the rules as described above, we can define transitions among configurations. 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 used. With any computation (halting or not) we associate a spike train, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0. The computation result is defined as the time interval between the first two spikes.

Denote by $N_2(\Pi)$ the set of numbers generated by Π, and by $N_2SNPSPA_m^n$ the family of all $N_2(\Pi)$ generated by SNP-SPA system, where system Π has at most m neurons and at most n rules in each neuron. When m or n is not restricted, it can be replaced by "*".

When SNP-SPA system works on accepting mode, it has no output neuron, and a spike sequence is received from the environment by input neuron. The received number n will be stored in a given neuron in the form of n spikes. When the computation halts, number n is said to be accepted by the system. Denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π, and by $N_{acc}SNPSPA_m^n$ the family of all $N_{acc}(\Pi)$ accepted by SNP-SPA system, where system Π has at most m neurons and at most n rules in each neuron.

3.2. Illustrative example

Fig. 2 provides an example, which is used to explain the working mechanism of SNP-SPA system. The system consists of four neurons labeled by A, B, C and D respectively. Initially, neurons σ_A and σ_D each contain a spike, and other neurons have no spike. The working mechanism can be illustrated as follows. Since at time $t = 1$ neuron σ_D has a spike, rule $a \to a$ is used to send the first spike to the environment. At the same time, with a spike in neuron σ_A, rule $a \to a; \pm 1(A, \{B, C\})$ is applied to create a new synapse between neurons σ_A and σ_B or between neurons σ_A and σ_C nondeterministically. Therefore, there are the following two cases.

(1) If the synapse between neurons σ_A and σ_C is created, then neuron σ_A sends a spike to neuron σ_C. And then the synapse between neurons σ_A and σ_C is deleted. At time $t = 2$, with a spike in neuron σ_C, rule $a \to a$ is applied to send a spike to neuron σ_D. At time $t = 3$, neuron σ_D applies rule $a \to a$ to send the second spike to the environment. Thus,

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

Table 1
The computation procedure of value "2".

Step	Neurons	Executing rules	*syn*
$t=1$	σ_A, a	$a \to a; \pm 1(A, \{B, C\})$	$syn \cup \{(A, C)\}$
	σ_D, a	$a \to a$	
$t=2$	σ_C, a	$a \to a$	*syn*
$t=3$	σ_D, a	$a \to a$	*syn*

Table 2
The computation procedure of value "4".

Step	Neurons	Executing rules	*syn*
$t=1$	σ_A, a	$a \to a; \pm 1(A, \{B, C\})$	$syn \cup \{(A, B)\}$
	σ_D, a	$a \to a$	
$t=2$	σ_B, a	$a \to \bar{a}$	*syn*
$t=3$	$\sigma_A, \bar{a}$	$\bar{a} \to a; \pm 1(A, \{C\})$	$syn \cup \{(A, C)\}$
$t=4$	σ_C, a	$a \to a$	*syn*
$t=5$	σ_D, a	$a \to a$	*syn*

the system halts. Therefore, the computation result is $3-1=2$. Table 1 provides the computation procedure of value "2".

(2) If the synapse between neurons σ_A and σ_B is generated, then neuron σ_A sends a spike to neuron σ_B. And then the synapse between neurons σ_A and σ_B is deleted. At time $t=2$, with a spike in neuron σ_B, rule $a \to \bar{a}$ is used to send an anti-spike to neuron σ_A. At time $t=3$, with an anti-spike in neuron σ_A, rule $\bar{a} \to a; \pm 1(A, \{C\})$ is applied to create a new synapse between neurons σ_A and σ_C, and a spike is sent to neuron σ_C. Then, the synapse between σ_A and σ_C is deleted. At time $t=4$, neuron σ_C applies rule $a \to a$ to send a spike to neuron σ_D. At time $t=5$, with a spike in neuron σ_D, rule $a \to a$ is used to send the second spike to the environment. Thus, the system halts. Therefore, the computation result is $5-1=4$. Table 2 provides the computation procedure of value "4".

4. Universality results

In this section, the universality of SNP-SPA systems as number accepting/generating devices is first discussed, and then a small universal SNP-SPA system is constructed for computing functions.

4.1. SNP-SPA systems as number generating device

Theorem 4.1. $N_2SNPSPA_*^2 = NRE$

Proof. Since conclusion $N_2SNPSPA_*^2 \subseteq NRE$ is straightforward, we prove inverse conclusion $NRE \subseteq N_2SNPSPA_*^2$. Assume that $M_1 = (m, H, l_0, l_h, I)$ is a register machine working on generating mode. Since M_1 can characterize the NRE, an SNP-SPA system Π_1 is constructed to simulate M_1. Assume that all registers different from register 1 are empty, and register 1 is not decremented during the computation. The system Π_1 consists of three modules: (i) ADD module, simulating ADD instruction of M_1, shown in Fig. 3; (ii) SUB module, simulating SUB instruction of M_1, shown in Fig. 4; (iii) OUTPUT module, used to export the computation result, shown in Fig. 5. Note that each of the three modules contains a number of neurons with plasticity rules.

For each register r in M_1, there is a neuron σ_r of Π_1 that is associated with it. The number of spikes in σ_r is coded: if register r contains the number $n \geq 0$, then σ_r has n spikes, and vice versa. Each instruction l in H corresponds to a neuron σ_l, and some auxiliary neurons are contained in these modules. Initially, only neuron σ_{l_0} contains a spike, indicating that system Π_1 starts the simulation of instruction l_0 of M_1. Note that in a regular register machine, l_0 is an ADD instruction. The simulation of instruction $l_i : (OP(r), l_j, l_k)$ (OP is ADD or SUB) is illustrated as follows: starting from activation of l_0, neuron σ_r operates according to semantics of OP, and then a spike is sent to one of σ_{l_j} and σ_{l_k}. When neuron σ_{l_h} receives a spike, this means that the computation of M_1 is completed, and at the same time the computation result stored in register 1 is outputted to the environment by OUTPUT module.

To prove that the register machine M_1 can be correctly simulated by Π_1, it will be described in detail how ADD and SUB instructions are simulated by ADD and SUB modules respectively, and how OUTPUT module exports the computation result.

(1) ADD module, simulating $l_i : (ADD(r), l_j, l_k)$

Fig. 3 shows ADD module of Π_1, which is used to simulate ADD instruction of M_1. Suppose that at time t, an ADD instruction $l_i : (ADD(r), l_j, l_k)$ is simulated, meaning that the neuron σ_{l_i} receives a spike. Thus, rule $a \to a$ is used to send a

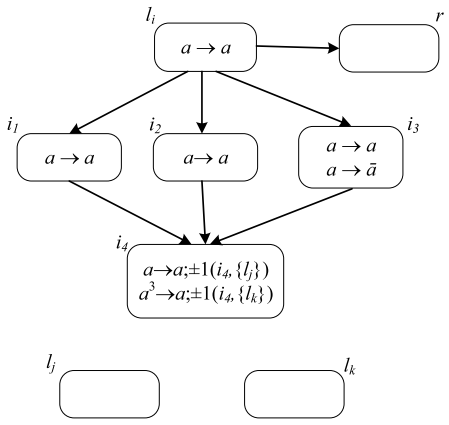

Fig. 3. A non-deterministic ADD module, which is used in generating mode.

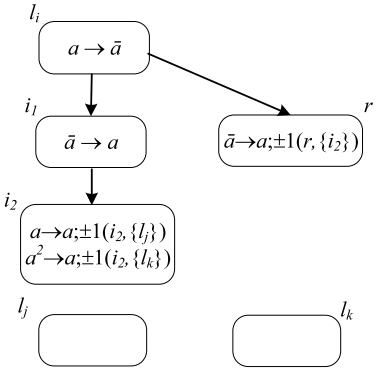

Fig. 4. SUB module.

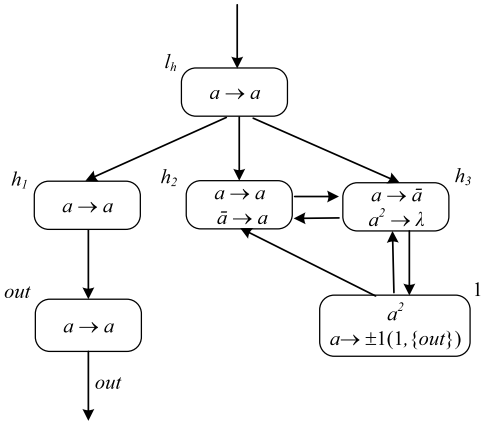

Fig. 5. OUTPUT module.

spike to neurons σ_{i_1}, σ_{i_2}, σ_{i_3} and σ_r, respectively. Since neuron σ_r receives a spike, register r is added by 1. At time $t+1$, since each of neurons σ_{i_1} and σ_{i_2} has a spike, they apply rule $a \to a$ to send a spike to neuron σ_{i_4}. At the same time, with a spike in neuron σ_{i_3}, two rules, $a \to a; \pm 1(i_4, \{l_j\})$ and $a^3 \to a; \pm 1(i_4, \{l_k\})$, can be applied, hence, one of them will be chosen nondeterministically. Therefore, there are the following two cases:

(i) At time $t+1$, if rule $a \to \bar{a}$ in neuron σ_{i_3} is applied, then it sends an anti-spike to neuron σ_{i_4}. Thus, neuron σ_{i_4} receives a spike (because a spike and an anti-spike are counteracted). At time $t+2$, with a spike in neuron σ_{i_4}, it applies rule $a \to a; \pm 1(i_4, \{l_j\})$ to create a synapse between neurons σ_{i_4} and σ_{l_j}, and a spike is sent to neuron σ_{l_j}. This means that the system is ready to simulate instruction l_j. At time $t+4$, the created synapse is deleted.

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

Table 3
The working procedure of OUTPUT module.

Step	Neurons	Executing rules	*syn*
t	σ_{l_h}, a	$a \to a$	*syn*
t+1	σ_{h_1}, a	$a \to a$	*syn*
	σ_{h_2}, a	$a \to a$	*syn*
	σ_{h_3}, a	$a \to \bar{a}$	*syn*
t+2	σ_{out}, a	$a \to a$	*syn*
	$\sigma_{h_2}, \bar{a}$	$\bar{a} \to a$	*syn*
	σ_{h_3}, a	$a \to \bar{a}$	*syn*
	σ_1, a^{n+1}	$\bar{a}a \to \lambda$	*syn*
...	...	...	...
t+n+1	σ_1, a	$a \to \pm 1(1, \{out\})$	$syn \cup \{(1, out)\}$
	$\sigma_{h_2}, \bar{a}$	$\bar{a} \to a$	*syn*
	σ_{h_3}, a	$a \to \bar{a}$ *syn*	
t+n+2	σ_{out}, a	$a \to a$	*syn*
	$\sigma_{h_2}, \bar{a}$	$\bar{a}a \to \lambda$	*syn*
	σ_{h_3}, a	$a^2 \to \lambda$	*syn*

(ii) At time $t+1$, if rule $a \to a$ in neuron σ_{i_3} is applied, then it sends a spike to neuron σ_{i_4}. Thus, neuron σ_{i_4} receives three spikes. At time $t+2$, rule $a^3 \to a; \pm 1(i_4, \{l_k\})$ in neuron σ_{i_4} is applied to create a synapse between neurons σ_{i_4} and σ_{l_k} and a spike is sent to neuron σ_{l_k}, meaning that the system starts the simulation of instruction l_k. At time $t+4$, the created synapse is deleted.

Hence, the ADD instruction is correctly simulated by the module.

(2) SUB module, simulating $l_i : (SUB(r), l_j, l_k)$

The SUB module is shown in Fig. 4, which is used to simulate SUB instruction of M_1. Suppose that at time t, a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is simulated. With a spike in neuron σ_{l_i}, it applies rule $a \to \bar{a}$ to send an anti-spike to neurons σ_{i_1} and σ_r. In the following, there are two cases based on whether the number of spikes in neuron σ_r is greater than 0:

(i) At time t, if neuron σ_r has the number of spikes, $n > 0$, then this indicates that the number stored in register r is n. Since neuron σ_r receives an anti-spike, implicit rule $a\bar{a} \to \lambda$ is applied to subtract a spike in it, indicating that the number stored in register r is decreased by 1. At time $t+1$, neuron σ_{i_1} sends a spike to neuron σ_{i_2}. At time $t+2$, with a spike in neuron σ_{i_2}, rule $a \to a; \pm 1(i_2, \{l_j\})$ is applied to create a synapse between neurons σ_{i_2} and σ_{l_j}, and a spike is sent to neuron σ_{l_j}. At time $t+3$, the created synapse is deleted. Once neuron σ_{l_j} receives a spike, this means that the system starts the simulation of instruction l_j.

(ii) At time t, if the number of spikes in neuron σ_r is zero, then this indicates that the number stored in register r is 0. At time $t+1$, rule $\bar{a} \to a; \pm 1(r, \{i_2\})$ in neuron σ_r is applied to create a synapse between neurons σ_r and σ_{i_2}, and it sends a spike to neuron σ_{i_2}. Meanwhile, rule $\bar{a} \to a$ in neuron σ_{i_1} is used to send a spike to neuron σ_{i_2}. At time $t+2$, the synapse between neurons σ_r and σ_{i_2} is deleted, and rule $a^2 \to a; \pm 1(i_2, \{l_k\})$ in neuron σ_{i_2} is applied to create a synapse between neurons σ_{i_2} and σ_{l_k} and a spike is sent to neuron σ_{l_k}. At time $t+3$, the synapse between neurons σ_{i_2} and σ_{l_k} is deleted. With a spike in neuron σ_{l_k}, this indicates that the system starts the simulation of instruction l_k.

It is worth noting that when the simulation ends, the SUB module is restored to the initial state, meaning that the SUB module can be used repeatedly during the simulation of register machine M_1. It can be seen from the above, the SUB module starts from neuron σ_{l_i} containing a spike, and at the end, neuron σ_{l_j} receives a spike (meaning that the value stored in register r is greater than 0) or neuron σ_{l_k} receives a spike (meaning that the value stored in register r is 0). This illustrates that the SUB instruction of the register machine is correctly simulated by the SUB module.

(3) OUTPUT module, outputting the computation result

Fig. 5 gives the OUTPUT module, which is used to export the computation result, and Table 3 provides a trace of its functioning. Suppose that neuron σ_{l_h} receives a spike at time t (indicating that register machine M_1 will halt and execute the halting instruction), and neuron σ_1 has $n+1$ spikes (indicating that the number contained in register 1 is n), where $n > 0$. With a spike in neuron σ_{l_h}, rule $a \to a$ is used to send a spike to neurons σ_{h_1}, σ_{h_2} and σ_{h_3}, respectively. Thus, neuron σ_1 contains $n+2$ spikes. At time $t+1$, neuron σ_{h_1} sends a spike to neuron σ_{out}. At the same time, neuron σ_{h_2} sends a spike to neuron σ_{h_3}; neuron σ_{h_3} sends an anti-spike to neurons σ_{h_2} and σ_1. At time $t+2$, neuron σ_{out} sends a spike to the environment. Since neuron σ_1 receives an anti-spike, its number of spikes is decreased by 1 due to implicit rule $\bar{a}a \to \lambda$. From time $t+2$ to time $t+n+1$, the above procedure on neurons σ_{h_2}, σ_{h_3} and σ_1 are respected constantly. At time $t+n+1$, with a spike in neuron σ_1, rule $a \to a; \pm 1(1, \{out\})$ is applied to create a synapse between neurons σ_1 and σ_{out}, and it sends a spike to neurons σ_{out}, σ_{h_2} and σ_{h_3}. At time $t+n+2$, with a spike in neuron σ_{out}, it sends a spike to the environment. At the same

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

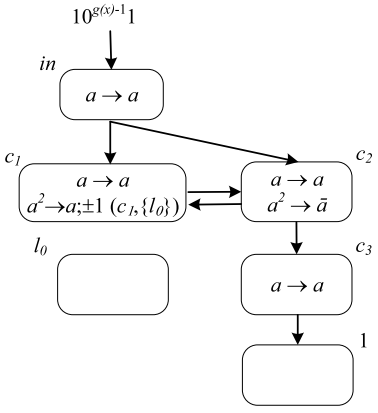

Fig. 6. INPUT module, which introduces the number $g(x)$ into neuron σ_1 from the environment.

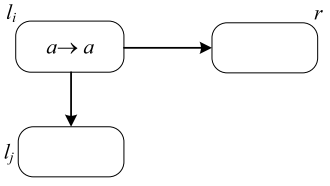

Fig. 7. A deterministic ADD module, which is used in accepting mode.

time, since neuron σ_{h_2} receives a spike, the anti-spike contained in it is consumed by implicit rule $\bar{a}a \to \lambda$; with two spikes in neuron σ_{h_3}, rule $a^2 \to \lambda$ to consume the two spikes. Note that the interval between the first two spikes sent by neuron σ_{out} is $(t+n+2)-(t+2)=n$, which is exactly the number of spikes contained in register 1 when the register machine halts.

From the discussion above, the system Π_1 correctly simulates the register machine M_1 working on generating mode, where each neuron contains two rules at most, and $N(M_1) = N_2(\Pi_1)$. Therefore, the theorem holds. □

4.2. SNP-SPA systems as number accepting device

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

Proof. The proof of this theorem is similar to Theorem 4.1, but only a deterministic register machine working on accepting mode, $M_2 = (m, H, l_0, l_h, I)$, is considered. To this end, an SNP-SPA system Π_2 is designed for the simulation of M_2, which contains an INPUT module receiving the spikes from the environment, shown in Fig. 6, a deterministic ADD module, shown in Fig. 7, and a SUB module that is the same as in generating mode. Initially, all neurons in the system are empty.

Fig. 6 shows the INPUT module, and Table 4 provides a trace of its functioning. A spike train $10^{g(x)-1}1$ is read from the environment by neuron σ_{in}. In the spike train, the time interval between two spikes is $(g(x)+1)-1 = g(x)$, which indicates that number $g(x)$ is accepted by the system. Assume that at time t, neuron σ_{in} receives the first spike from the environment. Thus, neuron σ_{in} sends a spike to neurons σ_{c_1} and σ_{c_2}, respectively. At time $t+1$, neurons σ_{c_1} and σ_{c_2} use rule $a \to a$ to send a spike to each other, and neuron σ_{c_2} sends a spike to neuron σ_{c_3}. At time $t+2$, neuron σ_{c_3} sends a spike to neuron σ_1. Thus, the neuron σ_1 receives a spike. At each step from step $t+2$ until neuron σ_{in} receives the second spike from the environment, neurons σ_{c_1} and σ_{c_2} send a spike to each other, and neuron σ_{c_3} sends a spike to neuron σ_1. At time $t+g(x)$, neuron σ_{in} receives the second spike from the environment. At time $t+g(x)+1$, with two spikes in neuron σ_{c_1}, rule $a^2 \to a; \pm 1(c_1, \{l_0\})$ is applied to create a synapse between neurons σ_{c_1} and σ_{l_0}, and it emits a spike to σ_{l_0}. Thus, neuron σ_{l_0} receives a spike, meaning that the system starts the simulation of instruction l_0. At the same time, neuron σ_{c_2} applies rule $a^2 \to \bar{a}$ to send to an anti-spike to neurons σ_{c_1} and σ_{c_3}. At time $t+g(x)+2$, neuron σ_{c_2} applies rule $a \to a$ to send to a spike to neurons σ_{c_1} and σ_{c_3}. At time $t+g(x)+3$, neurons σ_{c_1} and σ_{c_3} are restored to initial states because implicit rule $\bar{a}a \to \lambda$ is applied. Note that at each step from $t+2$ to $t+g(x)+1$, neuron σ_1 receives $g(x)$ spike from neuron σ_{c_3}, meaning that value $g(x)$ is introduced into neuron σ_1.

The ADD module is deterministic in accepting mode, shown in Fig. 7. When neuron σ_{l_i} receives a spike, $a \to a$ is used to send a spike to neurons σ_{l_j} and σ_r, respectively. Because neuron σ_r receives a spike, it simulates adding 1 to register r. Moreover, neuron σ_{l_j} receives a spike, meaning that it begins the simulation of instruction l_j. The SUB module is unchanged, shown in Fig. 4. The OUTPUT module is removed but neuron σ_{l_h} is remained in the system. When σ_{l_h} receives a spike, the computation halts, indicating that register machine M_2 begins to execute instruction l_h.

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

Table 4
The working procedure of INPUT module in Fig. 6.

Step	Neurons	Executing rules	*syn*
t	σ_{in}, a	$a \to a$	*syn*
t+1	σ_{c_1}, a σ_{c_2}, a	$a \to a$ $a \to a$	*syn*
t+2	σ_{c_1}, a σ_{c_2}, a σ_{c_3}, a	$a \to a$ $a \to a$ $a \to a$	*syn*
t+3	σ_{c_1}, a σ_{c_2}, a σ_{c_3}, a	$a \to a$ $a \to a$ $a \to a$	*syn*
...	...	...	... ...
t+g(x)	σ_{in}, a σ_{c_1}, a σ_{c_2}, a σ_{c_3}, a	$a \to a$ $a \to a$ $a \to a$ $a \to a$	*syn*
t+g(x)+1	σ_{c_1}, a^2 σ_{c_2}, a^2 σ_{c_3}, a	$a^2 \to \pm 1(c_1, \{l_0\})$ $a^2 \to \bar{a}$ $a \to a$	$syn \cup \{(c_1, l_0)\}$
t+g(x)+2	σ_{c_2}, a	$a \to a$	*syn*
t+g(x)+3	$\sigma_{c_3}, a, \bar{a}$	$\bar{a}a \to \lambda$	*syn*

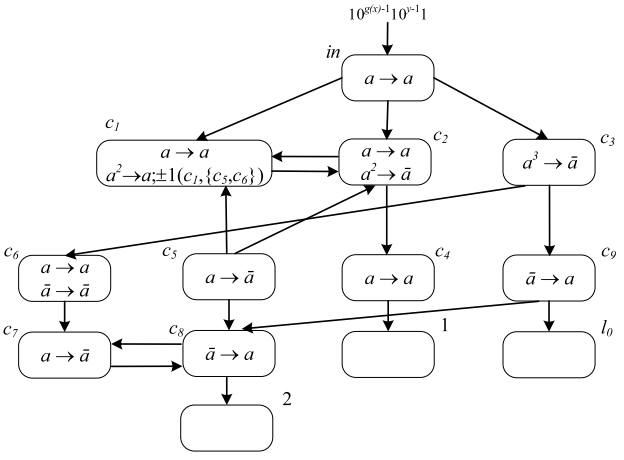

Fig. 8. INPUT module, which introduces the numbers $g(x)$ and y into neurons σ_1 and σ_2 from the environment respectively.

It can be found from the above description that when each neuron contains at most two rules, the SNP-SPA system correctly simulates the register machine M_2 working on accepting mode, and $N(M_2) = N_{acc}(\Pi_2)$. The theorem holds. □

4.3. SNP-SPA systems as function computing device

In the following, a small universal spiking neural P system with structural plasticity and anti-spikes is constructed to compute functions.

Theorem 4.3. *There is a small universal SNP-SPA system with 56 neurons for computing functions.*

Proof. We design an SNP-SPA system Π_3 to simulate the small universal register machine M'_u. The SNP-SPA system consists of four modules: INPUT, OUTPUT, ADD and SUB modules, where the last three modules are same as the previous modules, shown in Fig. 5, Fig. 7 and Fig. 4, respectively.

Fig. 8 shows the INPUT module, and Table 5 provides a trace of its functioning. In computing mode, the INPUT module is designed to receive spike train $10^{g(x)-1}10^{y-1}1$ from the environment and load $g(x)$ spikes into neuron σ_1 and y spikes into neuron σ_2 respectively. The working mechanism of this module is illustrated below. Assume that at time t, neuron σ_{in}

Table 5
The working procedure of INPUT module in Fig. 8.

Step	Neurons	Executing rules	*syn*
t	σ_{in}, a	$a \to a$	*syn*
t+1	σ_{c_1}, a	$a \to a$	*syn*
	σ_{c_2}, a	$a \to a$	
t+2	σ_{c_1}, a	$a \to a$	*syn*
	σ_{c_2}, a	$a \to a$	
	σ_{c_4}, a	$a \to a$	
t+3	σ_{c_1}, a	$a \to a$	*syn*
	σ_{c_2}, a	$a \to a$	
	σ_{c_4}, a	$a \to a$	
...	...	...	...
t+g(x)	σ_{in}, a	$a \to a$	*syn*
	σ_{c_1}, a	$a \to a$	
	σ_{c_2}, a	$a \to a$	
	σ_{c_4}, a	$a \to a$	
t+g(x)+1	σ_{c_1}, a^2	$a^2 \to a; \pm 1(c_1, \{c_5, c_6\})$	$syn \cup \{(c_1, c_5)\} \cup \{(c_1, c_6)\}$
	σ_{c_2}, a^2	$a^2 \to \bar{a}$	
	σ_{c_4}, a	$a \to a$	
t+g(x)+2	σ_{c_2}, a	$a \to a$	*syn*
	σ_{c_5}, a	$a \to \bar{a}$	
	σ_{c_6}, a	$a \to a$	
t+g(x)+3	$\sigma_{c_1}, a, \bar{a}^2$	$\bar{a}a \to \lambda$	*syn*
	$\sigma_{c_4}, a, \bar{a}$	$\bar{a}a \to \lambda$	
	σ_{c_7}, a	$a \to \bar{a}$	
	$\sigma_{c_8}, \bar{a}$	$\bar{a} \to a$	
t+g(x)+4	σ_{c_7}, a	$a \to \bar{a}$	*syn*
	$\sigma_{c_8}, \bar{a}$	$\bar{a} \to a$	
...	...	...	...
t+g(x)+y	σ_{in}, a	$a \to a$	*syn*
	σ_{c_7}, a	$a \to \bar{a}$	
	$\sigma_{c_8}, \bar{a}$	$\bar{a} \to a$	
t+g(x)+y+1	σ_{c_7}, a	$a \to \bar{a}$	*syn*
	$\sigma_{c_8}, \bar{a}$	$\bar{a} \to a$	
	$\sigma_{c_1}, a, \bar{a}$	$a\bar{a} \to \lambda$	
	$\sigma_{c_2}, a, \bar{a}$	$\bar{a}a \to \lambda$	
	σ_{c_3}, a^3	$a^3 \to \bar{a}$	
t+g(x)+y+2	$\sigma_{c_6}, \bar{a}$	$\bar{a} \to \bar{a}$	
	σ_{c_7}, a	$a \to \bar{a}$	
	$\sigma_{c_8}, \bar{a}$	$\bar{a} \to a$	
	$\sigma_{c_9}, \bar{a}$	$\bar{a} \to a$	
t+g(x)+y+3	$\sigma_{c_7}, a, \bar{a}$	$\bar{a}a \to \lambda$	*syn*
	$\sigma_{c_8}, a, \bar{a}$	$\bar{a}a \to \lambda$	

receives the first spike from the environment. Rule $a \to a$ in neuron σ_{in} is applied to send a spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_3}, respectively. At time $t+1$, rule $a \to a$ in neurons σ_{c_1} and σ_{c_2} is applied to send a spike to neurons σ_{c_4}, σ_{c_1} and σ_{c_2}. At each step from time $t+2$, neurons σ_{c_1} and σ_{c_2} each apply rule $a \to a$ to emit a spike to each other, and neuron σ_{c_4} applies rule $a \to a$ to send a spike to neuron σ_1 simultaneously.

At time $t+g(x)$, neuron σ_{in} fires by $a \to a$ after receiving the second spike from the environment. Thus, neurons σ_{c_1}, σ_{c_2} and σ_{c_3} each receive a spike, respectively. At time $t+g(x)+1$, with two spikes in neuron σ_{c_1}, rule $a^2 \to a; \pm 1(c_1, \{c_5, c_6\})$ is applied to create the synapse between σ_{c_1} and σ_{c_5} and the synapse between σ_{c_1} and σ_{c_6}, and it sends a spike to neurons σ_{c_5}, σ_{c_6} as well as neuron σ_{c_2}. Neuron σ_{c_2} applies rule $a^2 \to \bar{a}$ to send an anti-spike to neurons σ_{c_1} and σ_{c_4}. Then, the two synapses (between neurons σ_{c_1} and σ_{c_5} and between neurons σ_{c_1} and σ_{c_6}) are deleted. At time $t+g(x)+2$, with only one anti-spike in each of neurons σ_{c_1} and σ_{c_4}, they stop to send the spike to neurons σ_{c_2} and σ_1, respectively. Meanwhile, neuron σ_{c_2} applies rule $a \to a$ to a spike to neurons σ_{c_1} and σ_{c_4}, neuron σ_{c_5} applies rule $a \to \bar{a}$ to an anti-spike to neurons σ_{c_1}, σ_{c_2} and σ_{c_8}, and neuron σ_{c_6} applies rule $a \to a$ to a spike to neuron σ_{c_7}. Therefore, neuron σ_1 receives in total $(t+g(x)+2)-(t+3)+1 = g(x)$ spikes from neuron σ_{c_4}. At time $t+g(x)+3$, the implicit rule $a\bar{a} \to \lambda$ in neurons σ_{c_1} and σ_{c_4} is applied to consume a spike, rule $a \to \bar{a}$ in neuron σ_{c_7} is applied to send an anti-spike to neuron σ_{c_8}, and rule $\bar{a} \to a$

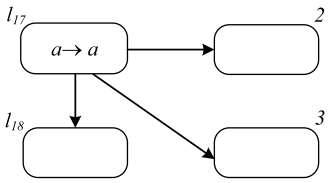

Fig. 9. The module simulating consecutive ADD-ADD instruction $l_{17} : (ADD(2), l_{21}), l_{21} : (ADD(3), l_{18})$.

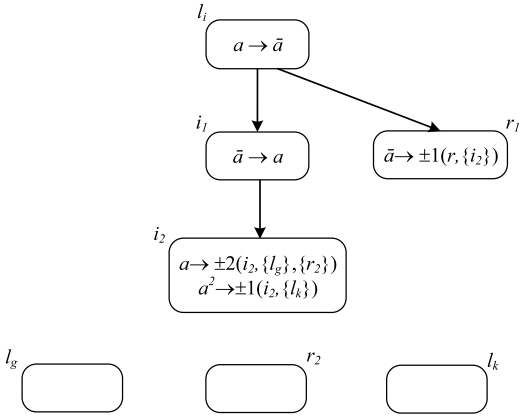

Fig. 10. The module simulating consecutive SUB-ADD-1 instructions $l_i : (SUB(r_1), l_j, l_k), l_l : (ADD(r_2), l_g)$.

in neuron σ_{c_8} is used to send a spike to neurons σ_2 and σ_{c_7}. At each step from this time, neurons σ_{c_7} and σ_{c_8} send a spike to each other, and neuron σ_{c_8} sends a spike to neuron σ_{c_2}.

At time $t + g(x) + y$, neurons σ_{c_1}, σ_{c_2} and σ_{c_3} receive a spike again after neuron σ_{in} receives the third spike from the environment. At time $t + g(x) + y + 1$, the implicit rule $a\bar{a} \to \lambda$ in neurons σ_{c_1} and σ_{c_2} is used to consume a spike, and rule $a^3 \to \bar{a}$ in neuron σ_{c_3} is applied to send an anti-spike to neurons σ_{c_5}, σ_{c_6} and σ_{c_9}. At time $t + g(x) + y + 2$, rule $\bar{a} \to \bar{a}$ in neuron σ_{c_6} is used to send an anti-spike to neuron σ_{c_7}, and rule $\bar{a} \to a$ in neuron σ_{c_9} is used to send a spike to neurons σ_{l_0} and σ_{c_8}. At time $t + g(x) + y + 3$, the implicit rule $a\bar{a} \to \lambda$ in neurons σ_{c_7} and σ_{c_8} is applied to consume a spike, hence, they stop to send the spike to each other, and neuron σ_{c_8} stops to send the spike to neuron σ_2. Therefore, neuron σ_2 receives $(t + g(x) + y + 3) - (t + g(x) + 4) + 1 = y$ spikes from neuron σ_{c_8}. With a spike in σ_{l_0}, the system is ready to simulate instruction l_0. The work of the INPUT module is finished.

So a total of 74 neurons are used in this system: 9 neurons for 9 registers; 25 neurons for instruction labels; each SUB module uses 2 auxiliary neurons, 28 in total; 10 neurons are used in the INPUT module, and the OUTPUT module uses 2 neurons.

Taking advantage of the particularities of the register machine M'_u, we can further reduce the number of neurons. First, let's observe that the sequence of two consecutive ADD instructions

$l_{17} : (ADD(2), l_{21}), l_{21} : (ADD(3), l_{18})$.

It can be simulated by the module from Fig. 9, and in this way we save the neuron $\sigma_{l_{21}}$.

Based on the number of spikes in register r_1 (associated with SUB instructions), the SUB and ADD instructions $l_i : (SUB(r_1), l_j, l_k), l_l : (ADD(r_2), l_g)$ can be divided into two cases. If the number of spikes in register r_1 is greater than 0, it means instruction $l_i = l_l$, module SUB-ADD-1 can simulate these two consecutive instructions as shown in Fig. 10. The neuron σ_{l_l} can be saved. There are 6 pairs of SUB-ADD-1 instructions in M'_u.

$l_0 : (SUB(1), l_1, l2), l_1 : (ADD(7), l_0)$;
$l_4 : (SUB(6), l_5, l3), l_5 : (ADD(5), l_6)$;
$l_6 : (SUB(7), l_7, l8), l_7 : (ADD(1), l_4)$;
$l_8 : (SUB(6), l_9, l0), l_9 : (ADD(6), l_{10})$;
$l_{14} : (SUB(5), l_{16}, l_{17}), l_{16} : (ADD(4), l_{11})$;
$l_{22} : (SUB(0), l_{23}, l'_h), l_{23} : (ADD(8), l_{22})$.

Therefore, we can save 6 neurons from the SUB-ADD-1 instructions.

If the number of spikes in register r_1 is 0, module SUB-ADD-2 $l_i : (SUB(3), l_{18}, l_{20}), l_{20} : (ADD(0), l_0)$ can simulate the two consecutive instructions, shown in Fig. 11. The neuron $\sigma_{l_{20}}$ can be saved.

In addition, in the module shown in Fig. 12, there is a common auxiliary neuron for all SUB instructions, which are associated with different registers. The groups of instructions $l_0, l_3, l_4, l_6, l_{10}, l_{13}, l_{15}$ and l_{19}, which address registers $1, 5, 6, 7, 4, 2, 3$ and 0 respectively, can save 7 neurons; and the group of instructions l_8, l_{11}, l_{18} and l_{22}, which address registers 6, 5, 4 and 0 respectively, can save 3 neurons.

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

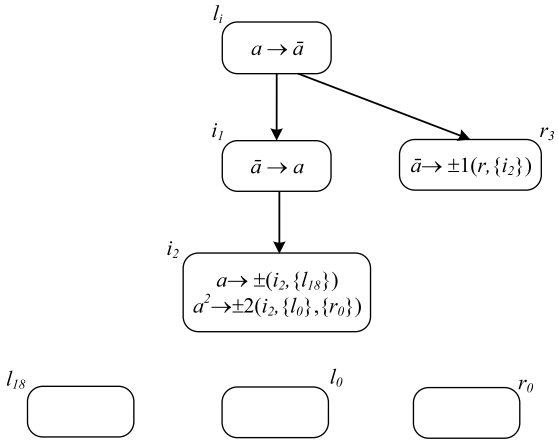

Fig. 11. The module simulating consecutive SUB-ADD-2 instructions $l_i : (SUB(3), l_{18}, l_{20}), l_{20} : (ADD(0), l_0)$.

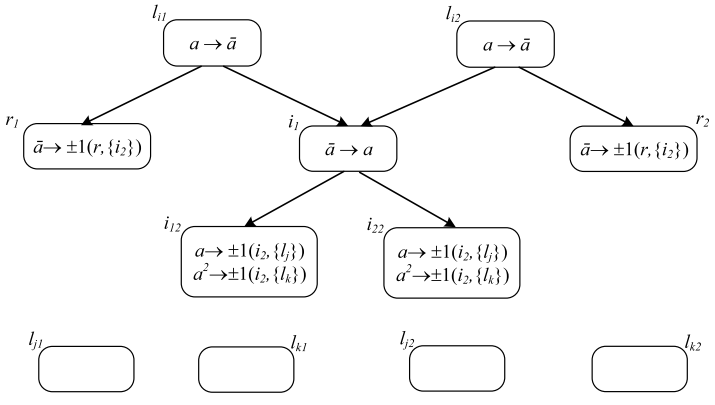

Fig. 12. The module simulating two SUB instructions with $r_1 \neq r_2$.

Table 6
The comparison of different computing models in the term of small number of computing units.

Computing models	Number of neurons
SNP-SPA systems	56
SNPA systems [21]	75
SNP-SP systems [48]	62
DTPN systems [31]	109
SNP systems [51]	84
SNP systems with standard rule [55]	7/4
WSNP systems [54]	48/45
SNQ P systems with two types of spikes [38]	49
SNQ P systems with one type of spikes [52]	181
SNQ P systems with one type of spikes [53]	14
SNP-MC systems [57]	57/39
SNP systems with anti-spikes [56]	24
Recurrent neural networks [50]	886

In total, we can save 18 neurons, so the number of neurons can be reduced from 74 to 56 in system Π_3. □

Theorem 4.3 shows a small number of computing units (i.e., neurons) for SNP-SPA systems as function computing devices to achieve universality. To further evaluate computational power of SNP-SPA systems, Table 6 provides the comparison result of the proposed variant with other computing models in the term of small number of computing units. The (proposed and compared) variants have different features or mechanisms. From Table 6, we can observe that the variants with different numbers of neurons achieve the universality for computing functions. Moreover, the improved results for some models were addressed, for example, SNQ P systems [38,52,53]. The proposed SNP-SPA systems possess two features (structural plasticity

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

and anti-spikes) and uses 56 neurons. Therefore, SNP-SPA systems need fewer neurons to achieve Turing universality for computing function.

5. Conclusions

This paper investigated a variant of SNP systems, spiking neural P systems with structural plasticity and anti-spikes (in short, SNP-SPA systems). The Turing universality of SNP-SPA systems as number generating/accepting devices was discussed, and then a small universal function computing device with 56 neurons was constructed. Table 6 shows that for a universal function device, SNP-SPA systems require fewer neurons compared with other models. This indicates that the fusion of the two features can reduce the computational unit.

SNP-SPA systems are a kind of distributed and parallel computing devices. Compared with the existing variants of SNP systems, it has two features: structural plasticity and a pair of spikes and anti-spikes. From the perspective of application, based on the two features, SNP-SPA systems are a distributed parallel computation model with dynamic directed graph and provide the better expression ability for modeling real-world problems. Therefore, our future work is to apply SNP-SPA systems to solve some real-world problems, for example, flexible manufacturing system and supervisory control problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors thank the anonymous reviewers for providing very insightful and constructive suggestions, which have helped to improve the presentation of this paper.

References

[1] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[2] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, New York, 2010.
[3] C. Martin-Vide, J. Pazos, G. Păun, A. Rodríguez-Patón, Tissue P systems, Theor. Comput. Sci. 296 (2) (2003) 295–326.
[4] R. Freund, Gh. Păun, M.J. Pérez-Jiménez, Tissue-like P systems with channel-states, Theor. Comput. Sci. 330 (1) (2005) 101–116.
[5] G. Păun, Membrane Computing: An Introduction, Springer-Verlag, Berlin, Germany, 2012.
[6] G. Păun, M.J. Pérez-Jiménez, Solving problems in a distributed way in membrane computing: DP systems, Int. J. Comput. Commun. Control 5 (2) (2010) 238–250.
[7] L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-Del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez, Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes, Fundam. Inform. 153 (1–2) (2017) 147–172.
[8] B. Song, L. Pan, The computational power of tissue-like P systems with promoters, Theor. Comput. Sci. 641 (2016) 43–52.
[9] L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, G. Vaszil, Variants of P colonies with very simple cell structure, Int. J. Comput. Commun. Control 4 (3) (2009) 224–233.
[10] X. Zhang, Y. Liu, B. Luo, L. Pan, Computational power of tissue P systems for generating control languages, Inf. Sci. 278 (10) (2014) 285–297.
[11] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, Inf. Sci. 304 (20) (2015) 80–91.
[12] H. Peng, J. Wang, P. Shi, M.J. Pérez-Jiménez, A. Riscos-Núñez, An extended membrane system with active membranes to solve automatic fuzzy clustering problems, Int. J. Neural Syst. 26 (3) (2016) 1650004.
[13] H. Peng, P. Shi, J. Wang, A. Riscos-Núñez, M.J. Pérez-Jiménez, Multiobjective fuzzy clustering approach based on tissue-like membrane systems, Knowl.-Based Syst. 125 (2017) 74–82.
[14] G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, Int. J. Neural Syst. 24 (5) (2014) 1440006.
[15] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, H. Peng, Membrane computing and image processing: a short survey, J. Membr. Comput. 1 (2019) 58–73.
[16] E. Sánchez-Karhunen, L. Valencia-Cabrera, Modelling complex market interactions using PDP systems, J. Membr. Comput. 1 (2019) 40–51.
[17] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2006) 279–308.
[18] G. Păun, Spiking neural P systems with astrocyte-like control, J. Univers. Comput. Sci. 13 (11) (2007) 1707–1721.
[19] L. Pan, J. Wang, H.J. Hoogeboom, Spiking neural P systems with astrocytes, Neural Comput. 24 (3) (2012) 805–825.
[20] L. Pan, G. Păun, Spiking neural P systems with anti-spikes, Int. J. Comput. Commun. Control 4 (3) (2009) 273–282.
[21] T. Song, Y. Jiang, X. Shi, X. Zeng, Small universal spiking neural P systems with anti-spikes, J. Comput. Theor. Nanosci. 10 (4) (2013) 999–1006.
[22] J. Wang, H.J. Hoogeboom, L. Pan, G. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with weights, Neural Comput. 22 (10) (2010) 2615–2646.
[23] X. Zeng, X. Zhang, T. Song, L. Pan, Spiking neural P systems with thresholds, Neural Comput. 26 (7) (2014) 1340–1361.
[24] T. Song, L. Pan, G. Păun, Spiking neural P systems with rules on synapses, Theor. Comput. Sci. 529 (2014) 82–95.
[25] T. Song, L. Pan, Spiking neural P systems with rules on synapses working in maximum spiking strategy, IEEE Trans. Nanobiosci. 14 (4) (2015) 465–477.
[26] H. Peng, R. Chen, J. Wang, X. Song, T. Wang, F. Yang, Z. Sun, Competitive spiking neural P systems with rules on synapses, IEEE Trans. Nanobiosci. 16 (8) (2017) 888–895.
[27] H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Lou, X. Huang, Spiking neural P systems with multiple channels, Neural Netw. 95 (2017) 66–71.
[28] X. Song, J. Wang, H. Peng, G. Ning, Z. Sun, T. Wang, F. Yang, Spiking neural P systems with multiple channels and anti-spikes, Biosystems 167–170 (2018) 13–19.
[29] T. Wu, A. Păun, Z. Zhang, L. Pan, Spiking neural P systems with polarizations, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2018) 3349–3360.
[30] H. Peng, J. Wang, Coupled neural P systems, IEEE Trans. Neural Netw. Learn. Syst. 30 (6) (2019) 1672–1682.
[31] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, Dynamic threshold neural P systems, Knowl.-Based Syst. 163 (2019) 875–884.
[32] F.G.C. Cabarle, H.N. Adorna, M. Jiang, X. Zeng, Spiking neural P systems with scheduled synapses, IEEE Trans. Nanobiosci. 16 (8) (2017) 792–801.

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034

[33] T. Song, A. Rodríguez-Patón, P. Zheng, X. Zeng, Spiking neural P systems with colored spikes, IEEE Trans. Cogn. Dev. Syst. 10 (4) (2017) 1106–1115.
[34] M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Ecegioglu, M. Ionescu, S. Woodworth, Asynchronous spiking neural P systems, Theor. Comput. Sci. 410 (24) (2009) 2352–2364.
[35] T. Song, L. Pan, Gh. Păun, Asynchronous spiking neural P systems with local synchronization, Inf. Sci. 219 (10) (2012) 197–207.
[36] T. Song, L. Pan, K. Jiang, B. Song, W. Chen, Normal forms for some classes of sequential spiking neural P systems, IEEE Trans. Nanobiosci. 12 (3) (2013) 255–264.
[37] X. Zhang, L. Pan, A. Păun, On the universality of axon P systems, IEEE Trans. Neural Netw. Learn. Syst. 26 (11) (2017) 2816–2829.
[38] L. Pan, Gh. Păun, G. Zhang, F. Neri, Spiking neural P systems with communication on request, Int. J. Neural Syst. 27 (8) (2017) 1750042.
[39] H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, Inf. Sci. 235 (20) (2013) 106–116.
[40] J. Wang, P. Shi, H. Peng, M.J. Pérez-Jiménez, T. Wang, Weighted fuzzy spiking neural P systems, IEEE Trans. Fuzzy Syst. 21 (2) (2013) 209–220.
[41] J. Wang, H. Peng, Adaptive fuzzy spiking neural P systems for fuzzy inference and learning, Int. J. Comput. Math. 90 (4) (2013) 857–868.
[42] 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 Trans. Power Syst. 30 (3) (2015) 1182–1194.
[43] H. Peng, J. Wang, P. Shi, M.J. Pérez-Jiménez, A. Riscos-Núñez, Fault diagnosis of power systems using fuzzy tissue-like P systems, Integr. Comput.-Aided Eng. 24 (4) (2017) 401–411.
[44] 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 Trans. Smart Grid 9 (5) (2018) 4777–4784.
[45] F.G.C. Cabarle, H.N. Adorna, M.J. Pérez-Jiménez, T. Song, Spiking neural P systems with structural plasticity, Neural Comput. Appl. 26 (8) (2015) 1905–1917.
[46] T. Song, L. Pan, A normal form of spiking neural P systems with structural plasticity, Int. J. Swarm Intell. 1 (4) (2015) 344–357.
[47] F.G.C. Cabarle, H.N. Adorna, M.J. Pérez-Jiménez, T. Song, Sequential spiking neural P systems with structural plasticity based on max/min spike number, Neural Comput. Appl. 27 (5) (2016) 1337–1347.
[48] F.G.C. Cabarle, R.T.A. de la Cruz, H.N. Adorna, M.D. Dimaano, F.T. Peña, X. Zeng, Small spiking neural P systems with structural plasticity, in: C. Graciani, A. Riscos-Núñez, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Enjoying Natural Computing, in: Lecture Notes in Computer Science, vol. 11270, 2018, pp. 45–56.
[49] I. Korec, Small universal register machines, Theor. Comput. Sci. 168 (2) (1996) 267–301.
[50] H.T. Siegelmann, E.D. Sontag, On the computational power of neural nets, J. Comput. Syst. Sci. 50 (1) (1995) 132–150.
[51] A. Păun, Gh. Păun, Small universal spiking neural P systems, Biosystems 90 (1) (2007) 48–60.
[52] T. Wu, F.-D. Bîlbîe, A. Păun, L. Pan, F. Neri, Simplified and yet Turing universal spiking neural p systems with communication on request, Int. J. Neural Syst. 28 (8) (2018) 1850013.
[53] T. Pan, X. Shi, Z. Zhang, F. Xu, A small universal spiking neural P system with communication on request, Neurocomputing 275 (2018) 1622–1628.
[54] X. Zeng, L. Pan, M.J. Pérez-Jiménez, Small universal simple spiking neural P systems with weights, Sci. China Inf. Sci. 57 (9) (2014) 1–11.
[55] T. Neary, Three small universal spiking neural P systems, Theor. Comput. Sci. 567 (2015) 2–20.
[56] S. Liu, K. Zhou, S. Zeng, H. Qi, X. Chen, An improvement of small universal spiking neural p systems with anti-spikes, in: BIC-TA 2016, no. 1, 2016, pp. 226–236.
[57] X. Song, H. Peng, J. Wang, G. Ning, T. Wang, Z. Sun, Y. Xia, On small universality of spiking neural p systems with multiple channels, in: International Conference on Membrane Computing, 2018, pp. 229–245.

Please cite this article in press as: Q. Yang et al., Spiking neural P systems with structural plasticity and anti-spikes, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.08.034