Neurocomputing 171 (2016) 1548–1555

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Spiking neural P systems with homogeneous neurons and synapses

Keqin Jiang [a], Wenli Chen [a], Yuzhou Zhang [a], Linqiang Pan [b,*]

[a] School of Computer and Information, Anqing Normal University, Anqing 246133, Anhui, China
[b] Key Laboratory of Image Information Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

ARTICLE INFO

Article history:
Received 14 February 2015
Received in revised form
27 July 2015
Accepted 30 July 2015
Communicated by A. Prieto
Available online 10 August 2015

Keywords:
Bio-inspired computing
Neural-like computational model
Membrane computing
P system
Spiking neural P system

ABSTRACT

Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons process information and communicate by means of spikes, where neurons are different in the sense that they can have different sets of spiking rules. In this work, we consider a variant of SN P systems with two restrictions: (1) all neurons contain only spikes (neurons are homogeneous in this case, since they can be considered as containers of spikes), while the spiking rules are moved on the synapses; (2) all synapses are homogeneous in the sense that each synapse has the same set of rules. These restrictions correspond to the fact that the SN P system consists of only one kind of neurons and one kind of synapses. The computational power of SN P systems with homogeneous neurons and synapses is investigated. Specifically, it is proved that such systems are Turing universal as both number generating and number accepting devices.

1. Introduction

Natural computing is a research field that investigates human-designed computing inspired by nature as well as computing taking place in nature [1,2]. Examples of natural computing include neural computation inspired by the functioning of the brain, evolutionary computation inspired by Darwinian evolution of species, swarm intelligence inspired by the behavior of groups of organisms, and artificial life systems inspired by the properties of natural life in general.

Membrane computing is a branch of natural computing inspired by the structure and the functioning of the living cells, as well as by the cooperation of cells in tissues, colonies of cells, and neural nets [3]. The computational models in the area of membrane computing are usually called *P systems*, which are a class of distributed and parallel computational devices. Many variants of P systems have been proved to be a rich framework for handling many problems related to computing [4–11]. For example, P systems can provide some new ideas for designing optimization algorithms to obtain approximate solutions to the intractable problems [12–18]. A general information about membrane computing can be found in [19,20], with up-to-date information available at membrane computing website: http://ppage.psystems.eu. The present work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short) [21].

SN P systems are inspired by the way in which neurons communicate by means of spikes (electrical impulses of identical shape). SN P systems use individual spikes allowing us to incorporate spatial and temporal information in computation. Neurons use spatial and temporal information of incoming spikes to encode their message to other neurons, where the number and timing of spikes matter. In the above sense, SN P systems fall into the third generation of neural network [22–25]. So, SN P systems provide a novel viewpoint to investigate spiking neural networks.

Briefly, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph, where neurons send signals (*spikes*, denoted by the symbol a in what follows) along *synapses* (arcs of the graph). Thus, the architecture of an SN P system is that of a tissue-like P system, with only one kind of objects present in the cells. Spikes evolve by means of *spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1$, $d \geq 0$. By means of a rule of this form, a neuron containing k spikes such that $a^k \in L(E)$, $k \geq c$, can consume c spikes and produce one spike after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. Spikes can also be removed from a neuron by rules of the form $a^s \to \lambda$ (called *forgetting rules*). One of the neurons is designated as the output neuron and its spikes are also sent to the environment. When a spike is emitted by the output neuron, we mark that time instance with 1; otherwise, we mark it with 0. This binary sequence is called the *spike train* of the system. The *result of a computation* is defined as

* Corresponding author.
E-mail addresses: jiangkq0519@163.com (K. Jiang), chenwl@aqtc.edu.cn (W. Chen), zhangyuzhou@aqtc.edu.cn (Y. Zhang), lqpan@mail.hust.edu.cn (L. Pan).

http://dx.doi.org/10.1016/j.neucom.2015.07.097
0925-2312/

the time distance between the first two spikes sent to the environment by the output neuron.

Many computational properties of SN P systems have been investigated. Most variants of SN P systems are rather powerful from a computational point of view. They are Turing universal (equivalent to Turing machines or other equivalent computing devices) as number generating/computing devices or accepting devices [26–29], language generators [30–33], and function computing devices [34–36]. Certain classes of SN P systems are also efficient, which can be used to solve computationally hard problems [37–40], combinatorial optimization problems [41] and to diagnose electric power systems [42]. In all these systems, spiking rules and forgetting rules are placed in neurons, and neurons can have different sets of rules.

In [43], a new variant of SN P systems, called SN P systems with rules on synapses, are considered. In these systems, neurons contain only spikes, while the spiking and forgetting rules are moved on the synapses. At any step, when the number of spikes in a neuron is "recognized" by a rule on a synapse leaving from that neuron, the rule is enabled. By applying the enabled rule, a number of spikes are consumed in the neuron and one spike is sent to the neuron at the end of the synapse.

The computational completeness is first proved, then two small universal SN P systems with rules on synapses for computing functions are constructed in [43]. As expected, placing the spiking and forgetting rules on synapses proves to be a powerful feature, both simpler proofs and smaller universal systems are obtained in comparison with the case when the rules are placed in the neurons.

Homogeneity appears in several computational devices in computer science, for example, classical cellular automata are composed of simple and homogeneous units. Consideration of homogeneousness in SN P systems has also a clear biological motivation. In the (human) brain, there exist about 100 billion neurons, and most of them are very similar to each other. They are "designed" by nature to have the same "set of rules", "working" in a uniform way to transform input into output. From an implementation point of view, homogeneity benefits mass-production, since a single set of elements has to be produced.

In this work, we consider SN P systems with rules on synapses under the restriction that each synapse has the same set of rules. In this case, neurons can be considered as containers of spikes, so neurons are *homogeneous*; synapses are also *homogeneous* in the sense that each synapse has the same set of rules. Hence, we call such systems *SN P systems with homogeneous neurons and synapses*. We investigate the computational power of such SN P systems. Specifically, it is proved that SN P systems with homogeneous neurons and synapses are Turing universal as both number generating and number accepting devices.

2. SN P systems with homogeneous neurons and synapses

In this section, we introduce a few notions and notations on formal language and automata theory, and the formal definition of SN P systems with homogeneous neurons and synapses. We refer readers to [44] for more details on formal language and automata theory, and to [20,21] for the motivation and basic concepts on SN P systems.

For an alphabet V, V^* denotes the set of all finite strings over V; the empty string is denoted by λ. The set of all non-empty strings over V, that is, $V^*-\{\lambda\}$, is denoted by V^+. When $V=\{a\}$ is a singleton, we simply write a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$, resp. The family of recursively enumerable languages is denoted by *RE* and the family of Turing computable sets of natural numbers is denoted by *NRE* (it is the family of length sets of languages in *RE*).

An *SN P system with homogeneous neurons and synapses* of degree $m\geq 1$ is a construct of the form

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

where

- $O=\{a\}$ is the singleton alphabet (a is called *spike*);
- $\sigma_1,\sigma_2,\ldots,\sigma_m$ are *neurons* of the form $\sigma_i=(n_i)$ with $1\leq i\leq m$, where $n_i\geq 0$ is the *initial number of spikes* contained in σ_i;
- syn is the set of synapses; each element in syn is a pair of the form $((i,j),R)$, where (i,j) indicates that there is a synapse connecting neurons σ_i and σ_j, with $i,j\in\{1,2,\ldots,m\}$, $i\neq j$, and R is a finite homogeneous set of rules of the following two forms:
  (1) $E/a^c\rightarrow a;d$, where E is a regular expression over O, $c\geq 1$ and $d\geq 0$;
  (2) $a^s\rightarrow\lambda$, for some $s\geq 1$, with the restriction that $a^s\notin L(E)$ for any rule $E/a^c\rightarrow a;d$ from R;
- $in,out\in\{1,2,\ldots,m\}$ indicate the *input* and *output* neurons, respectively.

Note that in SN P systems with homogeneous neurons and synapses, every synapse has the same set of rules. The rules of the form $E/a^c\rightarrow a;d$ are *firing* (we also say *spiking*) rules. They are applied as follows: if a rule $E/a^c\rightarrow a;d$ is on synapse (i,j) and neuron σ_i contains k spikes such that $a^k\in L(E)$, $k\geq c$, then the rule can be used. The application of this rule means that c spikes from neuron σ_i are consumed, and one spike is sent to neuron σ_j after a delay of d steps. If $d=0$, then the produced spike is sent to neuron σ_j immediately (in this case, the rule can be simply written as $E/a^c\rightarrow a$). If the rule is used in step t and $d\geq 1$, then in steps $t,t+1,\ldots,t+d-1$ the synapse is closed, that is, it cannot use any rule. In step $t+d$, the produced spike is sent to neuron σ_j, and synapse (i,j) can start to apply a rule at step $t+d+1$.

The rules of the form $a^s\rightarrow\lambda$ are *forgetting* rules. If neuron σ_i contains exactly s spikes, then the forgetting rule $a^s\rightarrow\lambda$ can be used, meaning that all s spikes are removed from σ_i. Note that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

A global clock is assumed, marking the time for all neurons and synapses. In each time unit, if a synapse (i,j) can use one of its rules, then a rule from R must be used. It is possible that two or more spiking rules can be used on a synapse at some moment, since two firing rules $E_1/a^{c_1}\rightarrow a;d_1$ and $E_2/a^{c_2}\rightarrow a;d_2$ can have $L(E_1)\cap L(E_2)\neq\emptyset$. In this case, the synapse will nondeterministically choose one of the enabled rules to use. Thus, the system works sequentially on each synapse (at most one rule from R can be used), and in parallel at the level of the system (if a synapse has at least one rule enabled, then it has to use a rule).

When several synapses starting from the same neuron have rules which can be applied, we work here with the restriction that all rules which are applied consume the same number of spikes from the given neuron.

The *configuration* of the system is described by both the number of spikes in each neuron and the number of steps to wait until each synapse becomes open (this number is zero if the corresponding synapse is already open). The *initial configuration* of the system is $C_0=\langle n_1,n_2,\ldots,n_m,0,0,\ldots,0\rangle$. Using the rules as described above, one can define *transitions* among configurations. A series 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 on any synapse. With any computation, halting or not, we associate a *spike train*, the binary sequence with occurrences of 1 indicating time instances when the output neuron sends a spike out of the system. The result of a successful computation is defined as usual in general SN P

systems, that is, for any spike train containing at least two spikes, the distance of first two spikes being emitted at steps $t1$, $t2$ is considered as the computation result, in the form of number $t2-t1$; we say that this number is generated/computed by the system.

SN P systems with homogeneous neurons and synapses can also work in the accepting mode: we start the computation from an initial configuration, and we introduce in the input neuron two spikes, at steps $t1$ and $t2$; the number $t2-t1$ is accepted by the system if the computation eventually halts.

In the following sections, SN P systems with homogeneous neurons and synapses are represented graphically. We use a circle with the initial number of spikes inside to represent a neuron and the directed edge associated with rules to represent the synapse. The input/output neuron has an incoming/outgoing arrow, respectively, suggesting their communication with the environment. Because each synapse has the same set of rules, the rules are omitted in the graphical representation.

We denote by $N_\alpha^{syn}HSNP$ the family of all sets of numbers generated/accepted by SN P systems with homogeneous neurons and synapses, where $\alpha \in \{gen, acc\}$ denoting the generating or accepting mode, and superscript *syn* means "rules on synapses".

3. Turing universality of systems working in the generating mode

In this section, we prove that SN P systems with homogeneous neurons and synapses can generate all recursively enumerable sets of numbers. So, they are Turing universal.

The following proofs are based on the simulation of register machines. A register machine is a construct $M=(m,H,l_0,l_h,I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

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

A register machine can work in the generating mode, and it generates (or computes) a number n in the following way: we start with all registers empty, we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels; if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize *NRE* (see, e.g., [45]).

In the accepting mode, deterministic register machines are universal. In this case, the ADD instructions are of the form l_i : (ADD(r), l_j). A deterministic register machine works in the following way: a number n is stored in the first register (all other registers are empty), the register machine starts computing with the instruction with label l_0; if the computation eventually halts, then the number n is accepted.

Theorem 3.1. $N_{gen}^{syn}HSNP = NRE$.

Proof. We have only to prove the inclusion $NRE \subseteq N_{gen}^{syn}HSNP$; the converse inclusion follows from Turing–Church Thesis, or it can be proved through a cumbersome but straightforward construction. To this aim, we use the characterization of *NRE* by means of register machines used in the generative mode. Let us consider a register machine $M=(m,H,l_0,l_h,I)$. Without any loss of generality, we may assume that in the halting configuration, all registers of M different from register 1 are empty, and that the register 1 is never decremented during the computation. In what follows, a specific SN P system with homogeneous neurons and synapses Π will be constructed to simulate the register machine M.

In the system Π, each synapse has the same set of rules

$$R=\{a\to\lambda,\ a^2\to a,\ a^3\to a,\ a^3\to a;1,\ a^4\to\lambda,$$
$$a^8/a\to a,\ a(a^5)^+/a^6\to a,\ a^4(a^5)^+/a^5\to a\}.$$

The system Π consists of three types of modules: ADD module, SUB module, and FIN module shown in Figs. 1, 2, 3, respectively. ADD module and SUB module are used to simulate the ADD and SUB instructions of M, respectively; FIN module is used to output a computational result.

In general, for each register r of M, a neuron σ_r is associated with: if register r contains the number $n\geq 0$, then this is encoded in the associated neuron σ_r by means of $5n$ spikes. For each label l_i of an instruction in M, a neuron σ_{l_i} is associated with. Initially, neuron σ_{l_0} associated with the initial instruction l_0 contains two spikes (which corresponds to the fact that M starts a computation by applying the instruction with label l_0). In general, if a neuron σ_{l_i} contains two spikes, the rule $a^2\to a$ on the synapses leaving σ_{l_i} is enabled and applied, which means that the instruction labeled

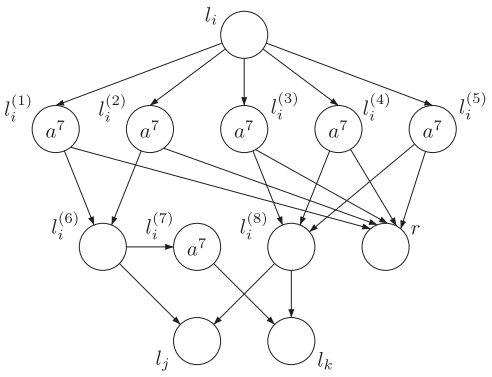

Fig. 1. Module ADD for simulating instruction l_i : (ADD(r), l_j, l_k).

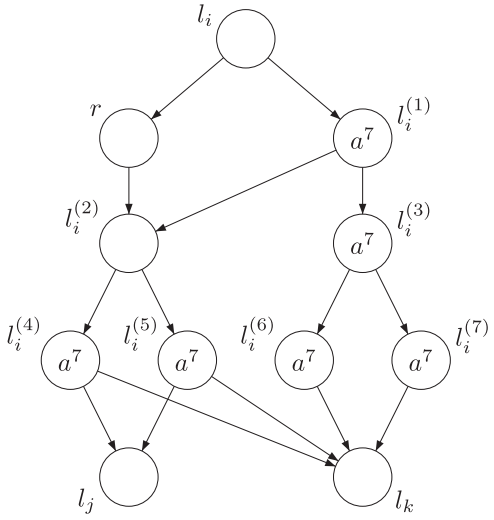

Fig. 2. Module SUB for simulating instruction l_i : (SUB(r), l_j, l_k).

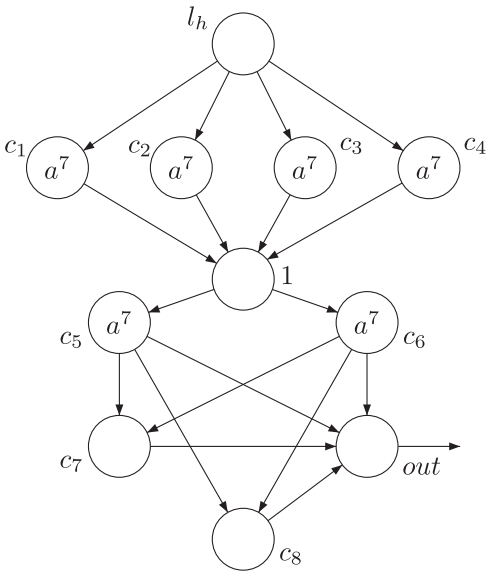

Fig. 3. Module FIN (outputting the result of computation).

with l_i starts to be simulated. When neuron σ_{l_h} contains two spikes, associated with the halting label of M, the computation in M is completely simulated in Π; then the output neuron will send two spikes to the environment, at an interval of time which corresponds to the number stored in register 1 of M.

In what follows, we show the mentioned modules in the graphical representation, and describe the work of these modules.

The module associated with an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ is given in Fig. 1. It works as follows. Assume that, at step t, neuron σ_{l_i} contains two spikes, then the five synapses leaving from this neuron fire by the rule $a^2 \to a$. At step $t+1$, each of neurons $\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$ contains 8 spikes, the synapses leaving from these five neurons fire by the rule $a^8/a \to a$. At step $t+2$, neuron σ_r receives 5 spikes (this simulates the increase of the number stored in register r by 1), neurons $\sigma_{l_i^{(6)}}$ and $\sigma_{l_i^{(8)}}$ receive 2 and 3 spikes, respectively. So, the two synapses leaving from neuron $\sigma_{l_i^{(6)}}$ fire by the rule $a^2 \to a$, and the two synapses leaving from neuron $\sigma_{l_i^{(8)}}$ fire by non-deterministically choosing one of the rules $a^3 \to a$ or $a^3 \to a; 1$. Thus, one of the neurons σ_{l_j} and σ_{l_k} gets two spikes and in this way the module associated with this neuron becomes active.

The evolution of the numbers of spikes in neurons of ADD module during the simulation of ADD instruction is shown in Tables 1 and 2, which correspond to the two cases in which neurons σ_{l_j} and σ_{l_k} are activated, respectively.

The module associated with a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is given in Fig. 2. Assume that, at step t, neuron σ_{l_i} contains two spikes, then synapses (l_i, r) and $(l_i, l_i^{(1)})$ fire by using the rule $a^2 \to a$, so each of σ_r and $\sigma_{l_i^{(1)}}$ receives one spike. In this way, σ_r contains $5n+1 (n \geq 0)$ spikes (corresponding to that the number stored in register r of M is n). If neuron σ_r contains one spike (i.e., $n=0$, corresponding to that the register r is empty), then this spike is removed by the forgetting rule $a \to \lambda$ on synapse $(r, l_i^{(2)})$, and finally σ_{l_k} gets two spikes, which means that the system starts to simulate the instruction l_k of register machine M. If neuron σ_r contains $5n+1$ $(n \geq 1)$ spikes (corresponding to that the number stored in the register r is not less than one), then synapse $(r, l_i^{(2)})$ fires by using the rule $a(a^5)^+/a^6 \to a$, consuming 6 spikes in neuron σ_r (which simulates that the number stored in register r is decreased by one), and sending one spike to neuron $\sigma_{l_i^{(2)}}$. At the same time, neuron $\sigma_{l_i^{(2)}}$ receives one spike from neuron $\sigma_{l_i^{(1)}}$. With two spikes in neuron $\sigma_{l_i^{(2)}}$, synapses $(l_i^{(2)}, l_i^{(4)})$ and $(l_i^{(2)}, l_i^{(5)})$ fire by using the rule $a^2 \to a$, and finally neuron σ_{l_j} gets two spikes through neurons $\sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$, which means that the system starts to simulate the instruction l_j of register machine M.

Note that there is no interference between ADD modules and SUB modules, but it is possible to have interference among SUB modules. Specifically, when simulating an instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$, neuron σ_r sends one spike to all neurons $\sigma_{l_s^{(2)}}$ from modules associated with instructions $l_s : (\mathtt{SUB}(r), l_u, l_v)$ (that is, subtracting from the same register r). However, no undesired effect appears: the spike arriving in neuron $\sigma_{l_s^{(2)}}$ $(s \neq i)$ is removed by the rule $a \to \lambda$ on synapse $(r, l_s^{(2)})$.

The evolution of the numbers of spikes of SUB module during the SUB instruction simulation is shown in Tables 3 and 4, corresponding to the two cases in which, at step t, the number stored in register r is $n, n > 0$ and the number stored in register r is 0, respectively.

The module associated with the HALT instruction $l_h : \mathtt{HALT}$ is given in Fig. 3. Assume that, at step t, neuron σ_{l_h} has two spikes, which means that the computation of register machine M halts and the system Π starts to output the computational result. Suppose that, at this moment, the number stored in register 1 of M is n, that is, the neuron associated with neuron σ_1 contains $5n$ spikes. With two spikes in neuron σ_{l_h}, the four synapses leaving from this neuron fire by the rule $a^2 \to a$. At step $t+1$, each of neurons $\sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}$ and σ_{c_4} contains 8 spikes, and the synapses leaving from these four neurons fire by the rule $a^8/a \to a$. At step

Table 1
The evolution of the numbers of spikes in neurons of ADD module during the simulation of ADD instruction with neuron σ_{l_j} finally activated.

Neuron	Step			
	t	$t+1$	$t+2$	$t+3$
σ_{l_i}	2	0	0	0
σ_r	$5n$	$5n$	$5n+5$	$5n+5$
$\sigma_{l_i^{(1)}}$	7	8	7	7
$\sigma_{l_i^{(2)}}$	7	8	7	7
$\sigma_{l_i^{(3)}}$	7	8	7	7
$\sigma_{l_i^{(4)}}$	7	8	7	7
$\sigma_{l_i^{(5)}}$	7	8	7	7
$\sigma_{l_i^{(6)}}$	0	0	2	0
$\sigma_{l_i^{(7)}}$	7	7	7	8
$\sigma_{l_i^{(8)}}$	0	0	3	0
σ_{l_j}	0	0	0	2
σ_{l_k}	0	0	0	1

Table 2
The evolution of the numbers of spikes in neurons of ADD module during the simulation of ADD instruction with neuron σ_{l_k} finally activated.

Neuron	Step				
	t	$t+1$	$t+2$	$t+3$	$t+4$
σ_{l_i}	2	0	0	0	0
σ_r	$5n$	$5n$	$5n+5$	$5n+5$	$5n+5$
$\sigma_{l_i^{(1)}}$	7	8	7	7	7
$\sigma_{l_i^{(2)}}$	7	8	7	7	7
$\sigma_{l_i^{(3)}}$	7	8	7	7	7
$\sigma_{l_i^{(4)}}$	7	8	7	7	7
$\sigma_{l_i^{(5)}}$	7	8	7	7	7
$\sigma_{l_i^{(6)}}$	0	0	2	0	0
$\sigma_{l_i^{(7)}}$	7	7	7	8	7
$\sigma_{l_i^{(8)}}$	0	0	3	3	0
σ_{l_j}	0	0	0	1	1
σ_{l_k}	0	0	0	0	2

Table 3
The evolution of the numbers of spikes in neurons of SUB module during the simulation of SUB instruction with the case that neuron σ_r has $n(n>0)$ spikes.

Neuron	Step				
	t	$t+1$	$t+2$	$t+3$	$t+4$
σ_{l_i}	2	0	0	0	0
σ_r	$5n$	$5n+1$	$5n-5$	$5n-5$	$5n-5$
$\sigma_{f_i^{(1)}}$	7	8	7	7	7
$\sigma_{f_i^{(2)}}$	0	0	2	0	0
$\sigma_{f_i^{(3)}}$	7	7	8	7	7
$\sigma_{f_i^{(4)}}$	7	7	7	8	7
$\sigma_{f_i^{(5)}}$	7	7	7	8	7
$\sigma_{f_i^{(6)}}$	7	7	7	8	7
$\sigma_{f_i^{(7)}}$	7	7	7	8	7
σ_{l_j}	0	0	0	0	2
σ_{l_k}	0	0	0	0	4

Table 4
The evolution of the numbers of spikes in neurons of SUB module during the simulation of SUB instruction with the case that the number of spikes in neuron σ_r is 0.

Neuron	Step				
	t	$t+1$	$t+2$	$t+3$	$t+4$
σ_{l_i}	2	0	0	0	0
σ_r	0	1	0	0	0
$\sigma_{f_i^{(1)}}$	7	8	7	7	7
$\sigma_{f_i^{(2)}}$	0	0	1	0	0
$\sigma_{f_i^{(3)}}$	7	7	8	7	7
$\sigma_{f_i^{(4)}}$	7	7	7	7	7
$\sigma_{f_i^{(5)}}$	7	7	7	7	7
$\sigma_{f_i^{(6)}}$	7	7	7	8	7
$\sigma_{f_i^{(7)}}$	7	7	7	8	7
σ_{l_j}	0	0	0	0	0
σ_{l_k}	0	0	0	0	2

Table 5
The evolution of the numbers of spikes in neurons of FIN module during the process outputting the result of computation.

Neuron	Step							
	t	$t+1$	$t+2$	$t+3$	$t+4$	...	$t+n+3$	$t+n+4$
σ_{l_h}	2	0	0	0	0	...	0	0
σ_1	$5n$	$5n$	$5n+4$	$5n-1$	$5n-6$	...	0	0
σ_{c_1}	7	8	7	7	7	...	7	7
σ_{c_2}	7	8	7	7	7	...	7	7
σ_{c_3}	7	8	7	7	7	...	7	7
σ_{c_4}	7	8	7	7	7	...	7	7
σ_{c_5}	7	7	7	8	8	...	7	7
σ_{c_6}	7	7	7	8	8	...	7	7
σ_{c_7}	0	0	0	0	2	...	2	0
σ_{c_8}	0	0	0	0	2	...	2	0
σ_{out}	0	0	0	0	2	...	4	2

$t+2$, neuron σ_1 contains $5n+4$ spikes, the two synapses leaving from neuron σ_1 fire by the rule $a^4(a^5)^+/a^5\to a$, five spikes are consumed in σ_1 (hence the rule $a^4(a^5)^+/a^5\to a$ can be used again in the next step until only four spikes remain in σ_1). At step $t+3$, neurons $\sigma_{c_5},\sigma_{c_6}$ and σ_1 contain 8, 8, $5n-1$ spikes, respectively. At step $t+4$, neuron σ_{out} receives two spikes from neurons $\sigma_{c_5},\sigma_{c_6}$, and the first spike is sent to the environment by the rule $a^2\to a$.

From step $t+5$ to step $t+n+3$, at each step, neuron σ_{out} receives four spikes from neurons $\sigma_{c_5},\sigma_{c_6},\sigma_{c_7}$ and σ_{c_8}, then the four spikes in neuron σ_{out} will be removed by the forgetting rule $a^4\to\lambda$. At step $t+n+4$, neuron σ_{out} receives two spikes from neurons σ_{c_7} and σ_{c_8}, then the second spike is sent to the environment. The interval between these two spikes sent out to the environment by the system is $(t+n+4)-(t+4)=n$, which means that the number computed by Π is n. Consequently, M and Π compute the same set of numbers.

The evolution of the numbers of spikes in neurons of FIN module during the process outputting the computational result is shown in Table 5.

From the above description of the modules and their work, it is clear that the register machine M is correctly simulated by the SN P system Π. Therefore, the theorem holds.□

4. Turing universality of systems working in the accepting mode

In this section, we investigate the computational power of SN P systems with homogeneous neurons and synapses working in the accepting mode. Specifically, we prove that SN P systems with homogeneous neurons and synapses working in the accepting mode are Turing universal.

Theorem 4.1. $N_{acc}^{syn}HSNP=NRE$.

Proof. We construct an SN P system with homogeneous neurons and synapses Π' to simulate a deterministic register machine $M=(m,H,l_0,l_h,I)$. The following proof is given by modifying the proof of Theorem 3.1. As in the proof of Theorem 3.1, in system Π', each synapse has the same set of rules:

$$R=\{a\to\lambda,\ a^2\to a,\ a^3\to a,\ a^3\to a;1,\ a^4\to\lambda,\\ a^8/a\to a,\ a(a^5)^+/a^6\to a,\ a^4(a^5)^+/a^5\to a\}.$$

The system Π' consists of an INPUT module, deterministic ADD modules and SUB modules. The INPUT module is shown in Fig. 4, which takes care of initializing the work of Π'.

After the first spike enters the input neuron σ_{in} at step t, neuron σ_{in} contains 8 spikes, then two synapses leaving from this neuron fire by the rule $a^8/a\to a$. At step $t+1$, each of neurons σ_{d_1} and σ_{d_2} contains 8 spikes, the synapses leaving from these two neurons fire by the rule $a^8/a\to a$. At step $t+2$, each of neurons $\sigma_{d_3},\sigma_{d_4},\sigma_{d_5}$ and σ_{d_6} receives two spikes from neurons σ_{d_1} and σ_{d_2}. With 7 spikes in neuron σ_{d_3}, the synapses leaving from neuron σ_{d_3} remain inactive (that is, no rules are enabled). With two spikes in

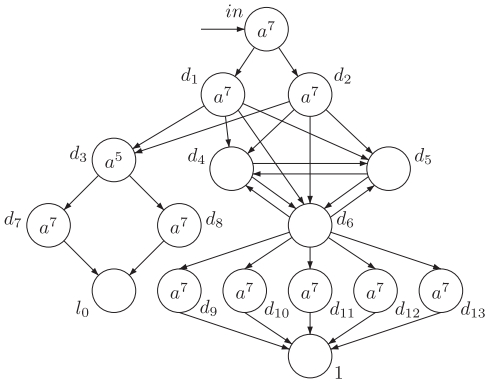

Fig. 4. Module INPUT (initializing the computation).

Table 6
The evolution of the numbers of spikes in neurons of INPUT module during the process initializing the computation (continued with Table 7).

Neuron	Step						
	t	$t+1$	$t+2$	$t+3$	$t+4$	…	$t+n$
σ_{in}	8	7	7	7	7	…	8
σ_1	0	0	0	0	5	…	$5n-15$
σ_{d_1}	7	8	7	7	7	…	7
σ_{d_2}	7	8	7	7	7	…	7
σ_{d_3}	5	5	7	7	7	…	7
σ_{d_4}	0	0	2	2	2	…	2
σ_{d_5}	0	0	2	2	2	…	2
σ_{d_6}	0	0	2	2	2	…	2
σ_{d_7}	7	7	7	7	7	…	7
σ_{d_8}	7	7	7	7	7	…	7
σ_{d_9}	7	7	7	8	8	…	8
$\sigma_{d_{10}}$	7	7	7	8	8	…	8
$\sigma_{d_{11}}$	7	7	7	8	8	…	8
$\sigma_{d_{12}}$	7	7	7	8	8	…	8
$\sigma_{d_{13}}$	7	7	7	8	8	…	8
σ_{l_0}	0	0	0	0	0	…	0

Table 7
The evolution of the numbers of spikes in neurons of INPUT module during the process initializing the computation (continuing Table 6).

Neuron	Step			
	$t+n+1$	$t+n+2$	$t+n+3$	$t+n+4$
σ_{in}	7	7	7	7
σ_1	$5n-10$	$5n-5$	$5n$	$5n$
σ_{d_1}	8	7	7	7
σ_{d_2}	8	7	7	7
σ_{d_3}	7	9	4	0
σ_{d_4}	2	4	0	0
σ_{d_5}	2	4	0	0
σ_{d_6}	2	4	0	0
σ_{d_7}	7	7	8	7
σ_{d_8}	7	7	8	7
σ_{d_9}	8	8	7	7
$\sigma_{d_{10}}$	8	8	7	7
$\sigma_{d_{11}}$	8	8	7	7
$\sigma_{d_{12}}$	8	8	7	7
$\sigma_{d_{13}}$	8	8	7	7
σ_{l_0}	0	0	0	2

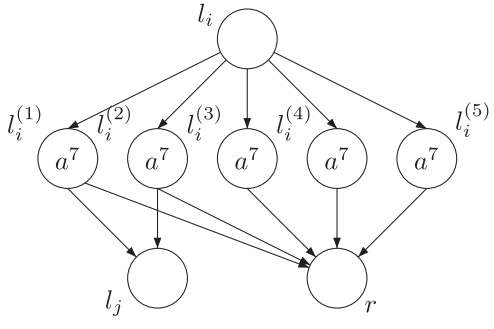

Fig. 5. Module ADD in the deterministic case.

neurons $\sigma_{d_4}, \sigma_{d_5}$ and σ_{d_6}, the synapses leaving from these three neurons fire by the rule $a^2 \to a$. At step $t+3$, each of neurons $\sigma_{d_9}, \sigma_{d_{10}}, \sigma_{d_{11}}, \sigma_{d_{12}}$ and $\sigma_{d_{13}}$ contains 8 spikes, the five synapses leaving from these neurons fire by the rule $a^8/a \to a$. In this way, the synapses leaving from neurons $\sigma_{d_4}, \sigma_{d_5}$ and σ_{d_6} will fire at each step, and the number of spikes in neuron σ_1 is increased by five at each step.

If at step $t+n$ ($n \geq 1$ is the number to be analyzed), the second spike enters neuron σ_{in} from the environment, then the two synapses leaving from neuron σ_{in} fire again by the rule $a^8/a \to a$. At step $t+n+1$, each of neurons σ_{d_1} and σ_{d_2} contains 8 spikes, the synapses leaving from these two neurons fire by the rule $a^8/a \to a$. At step $t+n+2$, each of neurons $\sigma_{d_3}, \sigma_{d_4}, \sigma_{d_5}$ and σ_{d_6} receives two spikes from neurons σ_{d_1} and σ_{d_2}. So, neuron σ_{d_3} contains 9 spikes, and each of neurons $\sigma_{d_4}, \sigma_{d_5}$ and σ_{d_6} contains four spikes. The four spikes in each of neurons $\sigma_{d_4}, \sigma_{d_5}$ and σ_{d_6} are removed by the forgetting rule $a^4 \to \lambda$. With 9 spikes in neuron σ_{d_3}, the synapses leaving from neuron σ_{d_3} fire by the rule $a^4(a^5)^+/a^5 \to a$, sending one spike to each of neurons σ_{d_7} and σ_{d_8}. So, each of neurons σ_{d_7} and σ_{d_8} contains 8 spikes, and the synapses leaving from these two neurons fire by the rule $a^8/a \to a$. In this way, neuron σ_{l_0} contains two spikes, which means that system Π' starts to simulate the initial instruction l_0 of M.

The evolution of the numbers of spikes in neurons of INPUT module during the process initializing the computation is shown in Tables 6 and 7.

The deterministic ADD module is shown in Fig. 5, and its functioning is rather clear, hence we omit the description of its work.

Module SUB remains unchanged as shown in Fig. 2, while module FIN is removed, with neuron σ_{l_h} remaining in the system, but without outgoing synapses. When neuron σ_{l_h} receives two spikes, it means that the computation of register machine M reaches instruction l_h and the system Π' halts.□

5. Conclusions and remarks

In this work, we investigated the computational power of a variant of SN P systems: SN P systems with homogeneous neurons and synapses. In such systems, there are only one kind of neurons and only one kind of synapses, so the computational power of systems depends on the way of the connection of neurons, that is, depends on the structure of systems. We proved that SN P systems with homogeneous neurons and synapses are Turing universal as both number generating and number accepting devices.

In the proof of Theorem 3.1, only the rule $a^3 \to a; 1$ has the feature of delay. It remains open whether SN P systems with homogeneous neurons and synapses are Turing universal when the feature of delay is not used.

As usual, it is worth investigating a series of normal forms for SN P systems with homogeneous neurons and synapses and constructing small universal SN P systems with homogeneous neurons and synapses. Taking into account the existence of many models in the SN P system framework, it would be interesting to simulate some variants through others, which will provide alternative methods to prove the computational completeness of these models. Another interesting problem is to explore the possibility of using SN P systems with homogeneous neurons and synapses as a framework for solving computationally hard problems such as Subset Sum problem and SAT problem.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61033003, 61370105, 91130034 and 61320106005), Ph.D. Programs Foundation of Ministry of Education of China (20120142130008), Anhui Provincial Natural Science Foundation (1408085MF131), Natural Science Research Project for Higher Education Institutions of Anhui Province (KJ2014A140), and

Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

[1] G. Rozenberg, T. Bäck, J.N. Kok (Eds.), Handbook of Natural Computing, Springer-Verlag, Berlin, 2012.
[2] G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: a comprehensive survey and new results, Inf. Sci. 279 (2014) 528–551.
[3] G. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[4] G. Zhang, J. Cheng, M. Gheorghe, Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, Appl. Soft Comput. 13 (3) (2013) 1528–1542.
[5] A. Ramanujan, K. Krithivasan, Control languages associated with tissue P systems, in: Lecture Notes in Computer Science, vol. 7956, 2013, pp. 186–197.
[6] V.P. Metta, K. Krithivasan, D. Grag, Modelling and analysis of spiking neural P systems with anti-spikes using Pnet lab, Nano Commun. Netw. 2 (2) (2011) 141–149.
[7] F.G.C. Cabarle, H. Adorna, M.A.Martínez-del. Amor, A spiking neural P system simulator based on CUDA, in: Lecture Notes in Computer Science, vol. 7184, 2012, pp. 87–103.
[8] F.G.C. Cabarle, H. Adorna, M.A. Martínez-del Amor, M.J. Pérez-Jiménez, Improving GPU simulations of spiking neural P systems, Romanian J. Inf. Sci. Technol. 15 (1) (2012) 5–20.
[9] H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy reasoning spiking neural P systems for fault diagnosis, Inf. Sci. 235 (2013) 106–116.
[10] 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.
[11] L. Xu, P. Jeavons, Simple neural-like P systems for maximal independent set selection, Neural Comput. 25 (6) (2013) 1642–1659.
[12] T.Y. Nishida, Membrane algorithms, in: Lecture Notes in Computer Science, vol. 3850, 2006, pp. 55–66.
[13] L. Huang, N. Wang, An optimization algorithm inspired by membrane computing, in: Lecture Notes in Computer Science, vol. 4222, 2006, pp. 49–52.
[14] L. Huang, X. He, N. Wang, Y. Xie, P systems based multi-objective optimization algorithm, Prog. Nat. Sci. 17 (4) (2007) 458–465.
[15] L. Huang, I.H. Suh, A. Abraham, Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants, Inf. Sci. 181 (2011) 2370–2391.
[16] G. Zhang, M. Gheorghe, C. Wu, A quantum-inspired evolutionary algorithm based on P systems for knapsack problem, Fund. Inf. 87 (1) (2008) 93–116.
[17] G. Zhang, M. Gheorghe, Y. Li, A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, Nat. Comput. 11 (4) (2012) 701–717.
[18] G. Zhang, J. Cheng, M. Gheorghe, Dynamic behavior analysis of membrane-inspired evolutionary algorithms, Int. J. Comput. Commun. Control 9 (2) (2014) 227–242.
[19] G. Păun, Membrane Computing—An Introduction, Springer-Verlag, Berlin, 2002.
[20] G. Păun, G. Rozenberg, A. Salomaa, (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, New York, 2010.
[21] M. Ionescu, G. Păun, T. Yokomori, Spiking neural P systems, Fund. Inf. 71 (2–3) (2006) 279–308.
[22] W. Maass, The Third Generation of Neural Network Models, Technische Universitat Gräz Syria, 1997.
[23] W. Gerstner, W.M. Kistler (Eds.), Spiking Neuron Models. Single Neurons, Populations, Plasticity, Cambridge University Press, Cambridge, 2002.
[24] W. Maass, Computing with spikes, Special Issue on Foundations of Information Processing of TELEMATIK 8 (1) (2002) 32–36.
[25] W. Maass, C.M. Bishop, (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, 1999.
[26] M. Cavaliere, O.H. Ibarra, G. Păun, O. Egecioglu, M. Ionescu, S. Woodworth, Asynchronous spiking neural P systems, Theor. Comput. Sci. 410 (24) (2009) 2352–2364.
[27] T. Song, L. Pan, G. Păun, Asynchronous spiking neural P systems with local synchronization, Inf. Sci. 219 (2013) 197–207.
[28] O.H. Ibarra, A. Păun, A. Rodríguez-Patón, Sequential SNP systems based on min/max spike number, Theor. Comput. Sci. 410 (2009) 2982–2991.
[29] 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.
[30] H. Chen, R. Freund, M. Ionescu, G. Păun, M.J. Pérez-Jiménez, On string languages generated by spiking neural P systems, Fundam. Inf. 75 (1–4) (2007) 141–162.
[31] H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, G. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with extended rules: universality and languages, Nat. Comput. 7 (2) (2008) 147–166.
[32] X. Zhang, X. Zeng, L. Pan, On languages generated by asynchronous spiking neural P systems, Theor. Comput. Sci. 410 (26) (2009) 2478–2488.
[33] K. Krithivasan, V.P. Metta, D. Garg, On string languages generated by spiking neural P systems with anti-spikes, Int. J. Found. Comput. Sci. 22 (1) (2011) 15–27.
[34] A. Păun, G. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 48–60.
[35] X. Zhang, X. Zeng, L. Pan, Smaller universal spiking neural P systems, Fundam. Inf. 87 (1) (2008) 117–136.
[36] L. Pan, X. Zeng, Small universal spiking neural P systems working in exhaustive mode, IEEE Trans. NanoBiosci. 10 (2) (2011) 99–105.
[37] T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang, Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources, Theor. Comput. Sci. 411 (25) (2010) 2345–2358.
[38] A. Leporati, G. Mauri, C. Zandron, G. Păun, M.J. Pérez-Jiménez, Uniform solutions to SAT and Subset Sum by spiking neural P systems, Nat. Comput. 8 (4) (2009) 681–702.
[39] L. Pan, G. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with neuron division and budding, Sci. China Inf. Sci. 54 (8) (2011) 1596–1607.
[40] P. Sosík, A. Rodríguez-Patón, L. Ciencialová, Polynomial complexity classes in spiking neural P systems, in: Lecture Notes in Computer Science, vol. 6501, 2011, pp. 348–360.
[41] 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) 1–16.
[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. to appear, http://dx.doi.org/10.1109/TPWRS.2014.2347699.
[43] T. Song, L. Pan, G. Păun, Spiking neural P systems with rules on synapses, Theor. Comput. Sci. 529 (2014) 82–95.
[44] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 1–3, Springer-Verlag, Berlin, 1997.
[45] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NJ, 1967.

Keqin Jiang was born in 1970. He is an Associate Professor of School of Computer and Information at Anqing Normal University, Anhui, China. He received Ph.D. degree from Huazhong University of Science and Technology in 2014. His current research interests include membrane computing, automata theory and its application.

Wenli Chen received the Bachelor' s degree from Anhui Normal University in 1996. Currently, she is a Lecturer of School of Computer and Information at Anqing Normal University, Anhui, China. Her current research interests include data mining, membrane computing, automata theory.

Yuzhou Zhang received the Bachelors degree from Anqing Normal University in 2001, and the M.S. degree from University of Science and Technology of China in 2011. Currently, he is an Associate Professor of School of Computer and Information at Anqing Normal University, Anhui, China. His current research interests include evolutionary computation, memetic algorithms, meta-heuristics and intelligent transportation.

Linqiang Pan received the Ph.D. degree from Nanjing University, China, in 2000. Since 2004, he is a Professor at Huazhong University of Science and Technology, Wuhan, China. His research interests include membrane computing, systems biology, and graph theory.