ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **17**, Number 2, 2014, 177–189

A Small Universal Spiking Neural P System with Cooperating Rules

Tao SONG, Linqiang PAN*

Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: lqpan@mail.hust.edu.cn

Abstract. Spiking neural P systems (shortly called SN P systems) are a class of distributed and parallel neural-like computing devices, which are inspired by the way of biological neurons communicating with each other by means of impulses/spikes. SN P systems with cooperating rules are a new variant of SN P systems, where each neuron has the same number of components and some components of a neuron can be empty. In a step of a computation, one component from each neuron is used, with the same label in all neurons; from these components, one rule is applied, in the way usual in SN P systems. In the terminating mode, adopted in this paper, after choosing a component of the neurons, this component is applied until no rule from this component, in any neuron, is enabled (we switch from a component to another one, nondeterministically chosen, when no rule of the component can be used, in any neuron of the system). In this work, we investigate how many neurons are needed to construct a Turing universal SN P system with cooperating rules as a number generator in terminating mode. Specifically, we construct a Turing universal SN P system having 8 neurons, which can generate/compute any set of Turing computable natural numbers. This result gives an answer to an open problem formulated in [V.P. Metta, S. Raghuraman, K. Krithivasan, CMC15, 267–282, 2014].

Key-words: bio-inspired computing, membrane computing, spiking neural P system, cooperating rule; Turing universality

*Corresponding author

1. Introduction

Spiking neural P systems (shortly called SN P systems) were introduced in [4] as a class of parallel and distributed computation models, which were abstracted from the neurophysiological behavior of neurons spiking and sending electrical impulses along axons to other neurons. In SN P systems, the processing elements are called *neurons* which are placed in the nodes of a directed graph, with the arcs representing the *synapses*. The content of each neuron consists of a number of copies of a single object type, called the *spike*. Every neuron may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses (also called *spikes*) which are accumulated at the target neurons. With the application of the forgetting rule, a predefined number of spikes will be removed from the neuron.

Many computational properties of SN P systems have been investigated (*e.g.* see [22]). SN P systems were proved to be Turing universal as number accepting or generating devices [4], language generators [1, 2], and function computing devices [10, 15, 19]. SN P systems with neuron division and budding can generate an exponential working space during the computation, thus provide a way to (theoretically) solve computationally hard problems in a feasible time [12].

Inspired by different biological facts, many variants of SN P systems have also been proposed, such as asynchronous SN P systems with local synchronization [26], SN P systems with astrocyte-like control [14, 21], SN P systems with anti-spikes [6, 13, 28], SN P systems with rules on synapses [27], some classes of homogenous SN P systems [30–32]. Most of the variants of SN P systems are Turing universal as number generating/accepting devices, language generating devices or function computing devices.

Finding small universal computing devices is a traditional research topic in computer science, whose aim is to construct universal computing devices (equivalent with the Turing machine) using less computation resources, such as small universal Turing machine [23], small universal register machine [5], small universal cellular automata [11]. The topic of finding small universal SN P systems was initialled by A. Păun and G. Păun in [19], where the computing resource refers to the number of neurons in the systems. It is obtained that as number generating devices 76 neurons are sufficient to achieve universality by using standard spiking rules; when using extended spiking rules, 50 neurons are sufficient. Following the research line, some smaller universal SN P systems have also been constructed, see [10, 33], where different techniques are proposed with the purpose of decreasing the number of neurons of Turing universal SN P systems. The number of neurons in Turing universal SN P systems can be reduced to 3 with using infinite rules in neurons [9]; if we use a finite number of rules in each neuron, the number of neurons in Turing universal SN P systems can be reduced to 10 [15]. For the variants of SN P systems, small universal systems were also constructed, *e.g.*, small universal SN P systems with anti-spikes are proposed in [25], small universal SN P systems with rules on synapse are constructed in [27], small universal sequential SN P systems [20], small universal spiking neural P systems working in exhaustive mode [16], small universal asynchronous SN P systems [17].

Recently, SN P systems with cooperating rules were proposed in [7], by introducing the concept of cooperation and distribution as known from the grammar systems [3] into SN P systems. In SN P systems with cooperating rules, each neuron has the same number of components, say q, labeled for each neuron with the same labels, $1, 2, \ldots, q$; some components of a neuron can be empty. In a step of a computation, one component from each neuron is used, with the same label in all neurons; from these components, one rule is applied, in the way usual in SN P systems. The active component can be switched from one to another during the computation, which depends on the different cooperation strategies (five cooperating strategies were proposed in [7]). In the terminating mode, adopted in this paper, after choosing a component j of the neurons, this component is applied until no rule from this component, in any neuron, is enabled (we switch from a component to another one, nondeterministically chosen, when no rule of the component can be used, in any neuron of the system). It was proved that asynchronous SN P systems with cooperating rules, and strongly sequential unbounded SN P systems with cooperating rules are Turing universal [7].

In this work, we investigate how many neurons are needed to construct Turing universal SN P system with cooperating rules, that is, how many neurons are needed to construct SN P systems with cooperating rules to simulate Turing Machines or equivalent computing devices. Specifically, we construct a Turing universal SN P system with cooperating rules having 8 neurons, which can generate/compute any set of Turing computable natural numbers. It is noted that the number of rules in any neuron of the Turing universal system is finite. Compared with the Turing universal SN P systems constructed in [15] (which has 10 neurons and finite rules in any neuron), the Turing universal system constructed in this work has less neurons (8 neurons), which implies that with the benefit of cooperating rules we can use less computing sources to achieve Turing universal SN P systems. As well, this result gives an answer to an open problem formulated in [7].

2. Spiking Neural P Systems with Cooperating Rules

Before we introduce SN P systems with cooperating rules, we recall some prerequisites. It is useful for readers to have some familiarity with basic elements of formal language theory, *e.g.*, from [24].

For an alphabet Σ, Σ^* denotes the set of all finite strings of symbols from Σ; the empty string is denoted by λ, and the set of all nonempty strings over Σ is denoted by Σ^+. When $\Sigma = \{a\}$ is a singleton, we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$.

A regular expression over an alphabet Σ is defined as follows: (i) λ and each $a \in \Sigma$ is a regular expression, (ii) if E_1, E_2 are regular expressions over Σ, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over Σ, and (iii) nothing else is a regular expression over Σ. 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 $a \in \Sigma$, (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 Σ. Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can

also be written as E^*.

We define here SN P systems with cooperating extended rules as well as the family of sets of numbers generated/computed by the systems.

An *SN P system with cooperating rules* of degree $m \geq 1$ is a construct as follows:

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

- $O = \{a\}$ is the singleton alphabet, a is called *spike*;
- $\Sigma = \{1, 2, \ldots, q\}$ is the set of labels of component;
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons* of the form $\sigma_i = (n_i, R_i)$ with $1 \leq i \leq m$, where
  (1) $n_i \in \mathbb{N}$ is the *initial numbers of spikes* contained in neuron σ_i, respectively;
  (2) $R_i = \bigcup_{l \in \Sigma} R_{il}$, where R_{il} is a finite set of *rules* (component l) containing rules of the following two forms:
    (a) $E/a^c \to a^p; d$, where E is a regular expression over O, $c \geq p, c \geq 1$ and $p, d \geq 0$;
    (b) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \to a^p; d$ from the same component;
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$, is the set of *synapses* between neurons;
- $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neuron, respectively.

Each neuron has the same number of components, say q, labelled by $1, 2, \ldots, q$, and each component may contain a set of rules; some components of a neuron can be empty. In a step of a computation, one component from each neuron is used, with the same label in all neurons; from these components, one rule is applied, in the way usual in SN P systems. After that another (not necessarily different) component of each neuron becomes active. At the beginning of the computation, a non-deterministically chosen component is active in each neuron. The way of switching active components is called a cooperation protocol. Series of cooperation protocols among the components in neurons of an SN P system have been considered in [7]. In the terminating mode, adopted in this paper, after choosing a component j of the neurons, this component is applied until no rule from this component, in any neuron, is enabled (we switch from a component to another one, nondeterministically chosen, when no rule of the component can be used, in any neuron of the system). For convenience, in what follows, the mode is not explicitly and repeatedly stated.

A rule $E/a^c \to a^p; d$ with $p \geq 1$ is called an *extended spiking rule*; if $p = 1$, the rule is called a *standard spiking rule*. The spiking rules are applied as follows. For $E/a^c \to a^p; d \in R_{il}$, if neuron σ_i contains k spikes such that $a^k \in L(E)$, $k \geq c$, and component l is active at that moment, then the rule is enabled; c spikes from neuron σ_i are consumed and p spikes are sent to each neuron σ_j with $(i, j) \in syn$ after a delay of d steps. If $d = 0$, then the p spikes are sent to the target neuron immediately. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, \ldots, t+d-1$ the synapse

cannot use any other rule. In step $t + d$, the p spikes are sent to neuron σ_j. If the rule is used in step t and $d \geq 1$, then in steps t, $t + 1$, ..., $t + d - 1$ the neuron is closed (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send several spikes along it, then these particular spikes are lost). If $L(E) = \{a^c\}$, then the rule can be written in the simplified form $a^c \to a^p; d$; if $d = 0$, then the rule can be simply written as $E/a^c \to a^p$.

In each time unit, if a neuron can use one of its rules, then a rule must be used. It is possible that there are more than one rule that can be used in a neuron at some moment, since two spiking rules, $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$, may have $L(E_1) \cap L(E_2) \neq \emptyset$. In this case, the neuron will non-deterministically choose one of the enabled rules to use.

A rule of the form $a^s \to \lambda$ is called a *forgetting rule*. When neuron σ_i contains exactly s spikes and the active component is l, then a forgetting rule $a^s \to \lambda \in R_{il}$ is enabled. By using it, $s \geq 1$ spikes are removed from the neuron.

The *configuration* of the system is described by both the number of spikes associated with each neuron and by the number of steps to wait until it becomes open (this number is zero if the neuron is already open). Thus, the *initial configuration* is $\langle n_1/0, n_2/0, \ldots, n_m/0\rangle$. In each neuron, at each step, if there is more than one rule enabled from the active component by its current contents, then only one of them (chosen non-deterministically) can fire. But the system as a whole evolves in a parallel and synchronizing way, at each step, all the neurons (that have an enabled rule) choose a rule from the active component and all of them fire at once. Using the rules in the above way, the system passes from one configuration to another configuration; such a step is called a *transition*.

A *computation* of system Π is a finite or infinite sequence of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. A computation halts when the system reaches a configuration where no rule can be used (i.e., the SN P system has halted). The *result of a computation* is defined as the number of spikes emitted by the output neuron when the system halts. The set of all numbers generated/computed in this way by system Π is denoted by $N_{tot}(\Pi)$ (the subscript *tot* indicates that the computation result is the total number of spikes emitting to the environment).

We denote by $N_{tot}SNP_mCR_p(\alpha, forg)$ the family of sets of numbers generated by SN P systems with cooperating rules of degree m, where the number of components in any neuron is p, $\alpha \in \{stan, extend\}$ indicates standard or extended spiking rules being used in any neuron, and forgetting rules are used. The indices m and p are replaced with $*$ when no bound is imposed on the respective parameter.

3. Small Universal SN P Systems with Cooperating Rules

In this section, we construct a Turing universal SN P systems with cooperating rules having 8 neurons. Since the related proof is based on the simulation of register machines, we recall the notion of register machine [8].

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:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to the register r and then go to one of the instructions with label l_j and l_k, non-deterministically chosen),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (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),
- $l_h : \texttt{HALT}$ (the halt instruction).

It is known that register machines generate all sets of numbers which are Turing computable, hence they characterize NRE (the family of Turing computable sets of numbers), even using register machines with only three registers [8].

Without loss of generality, it can be assumed that in the halting configuration, all registers different from the first (where the computation result is stored) one are empty, and that the first register is never decremented during the computation (its content is only added to).

We follow the following convention. When the computational power of two number generating devices D_1 and D_2 is compared, we use the convention that number zero is ignored; that is, $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$ (this corresponds to the usual practice of ignoring the empty string when comparing the power of two devices in language and automata theory).

Theorem 1. $N_{tot}SNP_8CR_2(extend, forg) = NRE$.

Proof. It is enough to prove the inclusion $NRE \subseteq N_{tot}SNP_8CR_2(extend, forg)$, since the converse inclusion is straightforward (or we can invoke for it from the Turing-Church thesis).

Let $M_u = (3, H, l_0, l_m, I)$ be a universal register machine with 3 registers (labeled by 0, 1, 2), where $H = \{l_0, l_1, l_2, \ldots, l_m\}$ is the set of instruction labels, l_0 is the start label and l_m is the halt label (assigned to instruction `HALT`). Without lose of generality, we assume that register 0 stores the computation result and is not subject to subtraction operations. In what follows, we shall construct an SN P systems with cooperating rules Π having 8 neurons to simulate the register machine M_u. The structure of system Π is given in Fig. 1, where each neuron has two components, *i.e.*, $\Sigma = \{1, 2\}$ and the spiking/forgetting rules are specified in Table 1.

The system Π consists of 8 neurons, labelled with *state*, s_1, s_2, d, $0, 1, 2$ and *out*. For each instruction l_i of M_u, a spiking rule in neuron σ_{state} is associated, and the rule is enabled if and only if neuron σ_{state} contains $T+i$ spikes, where $T = 11(m+1)$. For registers 0, 1 and 2 of register machine M_u, neurons σ_1, σ_2 and σ_3 are associated. The number stored in register i is represented by the number of spikes in neuron σ_i, $i = 0, 1, 2$. Specifically, if register 0 has number $n \geq 0$, there are $7n$ spikes in neuron σ_0; if register 1 has number $n \geq 0$, there are $8n$ spikes in neuron σ_1; if register 2 has

Table 1. The rules associated with neurons of system Π, where $T = 11(m+1)$

Neurons	The set of rules
σ_{s_1}	Component 1: $a^{T+2} \to a^T$ for ADD instructions acting on registers 0, 1, 2; Component 2: $a^{T+3}/a^{T+1} \to a^T$, $a^{T+2} \to a^T$, $a^{T+4}/a^{T+3} \to a^{T+1}$ for SUB instructions acting on registers 1 and 2;
σ_{s_2}	Component 1: $a^{T+2} \to a^T$ for ADD instructions acting on registers 0, 1, 2;
	Component 2: $a^{T+3}/a^{T+1} \to a^T$, $a^{T+2} \to a^T$, $a^{T+4}/a^{T+3} \to a^{T+1}$ for SUB instructions acting on registers 1 and 2;
σ_0	Component 1: $a^8(a^7)^*/a^8 \to a$, $a^9(a^7)^*/a^9 \to a$ for ADD instructions on registers 1 and 2; $a^3(a^7)^*/a^3 \to a$, $a^4(a^7)^*/a^4 \to a$ for SUB instructions acting on register 1; $a^5(a^7)^*/a^5 \to a$, $a^4(a^7)^*/a^4 \to a$ for SUB instructions acting on register 2; $a^{13}(a^7)^+/a^7 \to a^7$, $a^{13} \to a^7$; Component 2: $\emptyset$
σ_1	Component 1: $a^7(a^8)^*/a^7 \to a$, $a^9(a^8)^*/a^9 \to a$ for ADD instructions on registers 0 and 2; $a^3(a^8)^+/a^{11} \to a^3$, $a^3 \to a$, $a^4(a^8)^*/a^4 \to a^2$ for SUB instructions acting on register 1; $a^5(a^8)^*/a^5 \to a$, $a^4(a^8)^*/a^4 \to a^3$ for SUB instructions acting on register 2; Component 2: $\emptyset$
σ_2	Component 1: $a^7(a^9)^*/a^5 \to a$, $a^8(a^9)^*/a^7 \to a$ for ADD instructions on registers 0 and 1; $a^3(a^9)^*/a^3 \to a$, $a^4(a^9)^*/a^4 \to a^3$ for SUB instructions acting on register 1; $a^5(a^9)^+/a^{14} \to a^3$, $a^5 \to a$, $a^4(a^8)^*/a^4 \to a^2$ for SUB instructions acting on register 2; Component 2: $\emptyset$
σ_d	Component 1: $a^2 \to a^2$, $a^3 \to a^3$, $a^5 \to a^4$, $a^6 \to \lambda$, $a^7 \to \lambda$ Component 2: $\emptyset$
σ_{out}	Component 1: $a \to \lambda$, $a^7 \to a$; Component 2: $\emptyset$
σ_{state}	Component 1: $a^{T+i}/a^{T+i-j} \to a^7$, $a^{T+i}/a^{T+i-k} \to a^7$ for each ADD instruction $l_i : (\texttt{ADD}(0), l_j, l_k)$; $a^{T+i}/a^{T+i-j} \to a^8$, $a^{T+i}/a^{T+i-k} \to a^8$ for each ADD instruction $l_i : (\texttt{ADD}(1), l_j, l_k)$; $a^{T+i}/a^{T+i-j} \to a^9$, $a^{T+i}/a^{T+i-k} \to a^9$ for each ADD instruction $l_i : (\texttt{ADD}(2), l_j, l_k)$; $a^{T+i}/a^{i+1} \to a^3$, $a^{2T+i-1}/a^{T+i-j-1} \to a^4$, $a^{2T+i}/a^{T+i-k} \to a^4$ for each SUB instruction $l_i : (\texttt{SUB}(1), l_j, l_k)$ $a^{T+i}/a^{i+1} \to a^5$, $a^{2T+i-1}/a^{T+i-j-1} \to a^4$, $a^{2T+i}/a^{T+i-k} \to a^4$ for each SUB instruction $l_i : (\texttt{SUB}(2), l_j, l_k)$ $a^{T+m} \to a^6$ for halt instruction $l_m : \texttt{HALT}$. Component 2: $\emptyset$

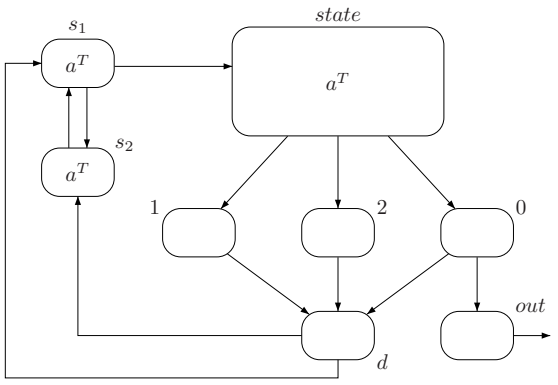

Fig. 1 The structure of system Π, where $T = 11(m + 1) + 1$.

number $n \geq 0$, there are $9n$ spikes in neuron σ_2. Neuron σ_{out} has an outgoing synapse to the environment outputting the computation result of the system. Neurons σ_{s_1} and σ_{s_2} are used to supplement the number of spikes consumed from neuron σ_{state} during the computation. The neurons in system Π has two components, labeled by 1 and 2. All neurons are associated with empty set of rules in their component 2, except that neurons σ_{s_1} and σ_{s_2} have rules in its component 2 (such rules can be used only when SUB instructions are simulated).

Initially, all neurons contain no spike with the exception that neurons σ_{state} has T spikes (it means the system starts to simulate instruction l_0), and each of neurons σ_{s_1},σ_{s_2} has T spikes. When neuron σ_{state} has $T+i$ spikes, system Π starts to simulate an instruction $l_i : (\texttt{OP}(r), l_j, l_k)$: starting with firing neuron σ_{state}, operating neuron σ_r as requested by OP, and the number of spikes in neuron σ_{state} becomes $T + j$ or $T + k$, which means that system Π starts to simulate instruction l_j or l_k. When neuron σ_{state} has $T + m$ spikes, it indicates that a computation in M_u is completely simulated in Π. The number of spikes emitted to the environment from neuron σ_{out} is exactly the number stored in register 0 of M_u. We can find that the function of neuron σ_{state} is somewhat similar with "the finite set of states" in Turing machine, that is why we use the label *state* for this neuron.

Simulation of ADD instructions

Assume that the system is at a step starting to simulate an ADD instruction $l_i : (\texttt{ADD}(0), l_j, l_k)$; that is, neuron σ_{state} contains $T+i$ spikes. So, the rules $a^{T+i}/a^{T+i-j} \rightarrow a^7$ and $a^{T+i}/a^{T+i-k} \rightarrow a^7$ are enabled, and one of them is non-deterministically chosen and applied.

If rule $a^{T+i}/a^{T+i-j} \rightarrow a^7$ is applied, neuron σ_{state} ends with $(T+i)-(T+i-j) = j$ spikes and sends 7 spikes to each of neurons σ_0, σ_1 and σ_2, respectively. The number of spikes in neuron σ_0 is increased by 7, which simulates the number stored in register 0 is added by one. By receiving 7 spikes, neurons σ_1 and σ_2 will fire by using rule

$a^7(a^8)^*/a^7 \to a$ and $a^7(a^9)^*/a^7 \to a$, sending two spikes to neuron σ_d. Neuron σ_d fires by rule $a^2 \to a^2$, sending 2 spikes to each of neurons σ_{s_1} and σ_{s_2}. Having $T+2$ spikes in neurons σ_{s_1} and σ_{s_2}, they fire by rule $a^{T+2} \to a^T$ sending T spikes to neuron σ_{state}, and the number of spikes in each of neuron σ_{s_1} and σ_{s_2} returns to T. In this way, the number of spikes in neuron σ_{state} becomes $T+j$, which means the system starts to simulate instruction l_j of machine M_u.

If rule $a^{T+i}/a^{T+i-k} \to a^7$ is applied, then neuron σ_{state} ends with $(T+i)-(T+i-k)=k$ spikes, and the number of spikes in neuron σ_0 is increased by 7, which simulates that the number stored in register 0 is added by one. Similar with the above case, after neurons σ_{s_1} and σ_{s_2} fire, the number of spikes in neuron σ_{state} becomes $T+k$, which means the system starts to simulate instruction l_k of machine M_u.

When system Π simulates an ADD instruction $l_i : (\texttt{ADD}(1), l_j, l_k)$, neuron σ_{state} contains $T+i$ spikes. So, one of the rules $a^{T+i}/a^{T+i-j} \to a^8$ and $a^{T+i}/a^{T+i-k} \to a^8$ is non-deterministically chosen and applied. The number of spikes in neuron σ_1 is increased by 8 (simulating the number stored in register 1 is added by one), and the number of spikes in neuron σ_{state} becomes $T+j$ or $T+k$, which means system Π non-deterministically starts to simulate instruction l_j or l_k of M_u.

Similarly, when system Π simulates an ADD instruction $l_i : (\texttt{ADD}(2), l_j, l_k)$, neuron σ_{state} fires by using one of the rules $a^{T+i}/a^{T+i-j} \to a^9$ and $a^{T+i}/a^{T+i-k} \to a^9$ non-deterministically. The number of spikes in neuron σ_2 is increased by 9 (simulating the number stored in register 2 is added by one); and the number of spikes in neuron σ_{state} becomes $T+j$ or $T+k$, which means system Π non-deterministically starts to simulate instruction l_j or l_k of M_u.

The simulation of ADD instruction is correct: system Π starts with neuron σ_{state} containing $T+i$ spikes; by firing neuron σ_{state}, the number of spikes in neuron σ_r is increased by $f(r)$ simulating the number stored in register r is increased by one, where $r = 0, 1, 2$ and $f(1) = 7$, $f(2) = 8$, $f(3) = 9$; and the system non-deterministically passes to simulate instruction l_j or l_k of M_u. In simulations of ADD instructions, only rules from component 1 of the involved neurons are applied; that is, during the simulation process, no component switching occurs.

Simulation of SUB instructions

Without loss of generality, we suppose that SUB instructions act only on register 1 and 2; that is, register 0 is not subject to SUB instructions, where the computation result is stored. Assume that the system is at a step simulating a SUB instruction $l_i : (\texttt{SUB}(1), l_j, l_k)$. At this moment, neuron σ_{state} contains $T+i$ spikes. So, by rule $a^{T+i}/a^{i+1} \to a^3$, neuron σ_{state} fires, sending 3 spikes to each of neurons σ_0, σ_1 and σ_2. By receiving 3 spikes, neurons σ_0 and σ_2 fire by rules $a^3(a^7)^*/a^3 \to a$ and $a^3(a^9)^*/a^3 \to a$, sending two spikes to neuron σ_d and one spike to neuron σ_{out}. The spike in neuron σ_{out} is removed by forgetting rule $a \to \lambda$. In neuron σ_1, there are the following two cases.

- If there is no spike in neuron σ_1 (corresponding to the fact that the number stored in register 1 is 0), then by receiving 3 spikes from neuron σ_{state}, rule $a^3 \to a$ in neuron σ_1 is applied, sending one spike to neuron σ_d. In this way,

neuron σ_d accumulates 3 spikes inside. It fires by rule $a^3 \to a^3$ sending 3 spikes to each of neurons σ_{s_1} and σ_{s_2}. At that moment, no rule from component 1 in the neurons of the systems can be used, so the system switches to apply rules from component 2 (due to the switching strategy among components is the terminating mode). In this way, neurons σ_{s_1} and σ_{s_2} fire by rule $a^{T+3}/a^{T+1} \to a^T$, sending T spikes to neuron σ_{state}, and the two neurons end with $T+2$ spikes inside. With $T+2$ spikes in neurons σ_{s_1} and σ_{s_2}, rule $a^{T+2} \to a^T$ from component 2 is applied, sending T spikes to each of neurons σ_{state}, σ_{s_1} and σ_{s_2}. The numbers of spikes in neuron σ_{s_1} and σ_{s_2} return to T, and neuron σ_{state} accumulates $2T+i-1$ spikes. In the next step, component 1 is active (since no rule from component 2 can be used), and neuron σ_{state} fires by rule $a^{2T+i-1}/a^{T+i-j-1} \to a^4$, sending 4 spikes to each of neurons σ_0, σ_1 and σ_2. Then, neurons σ_0, σ_1 and σ_2 fire sending 6 spikes to neuron σ_d. Neuron σ_d removes the 6 spikes by forgetting rule one step later, meanwhile neuron σ_{state} ends with $(2T+i-1)-(T+i-j-1) = T+j$ spikes inside, which simulates system Π starts to simulate instruction l_j of M_u.

- If there are $8n$ spikes in neuron σ_1 with $n > 0$ (corresponding to the fact that the number stored in register 1 is $n > 0$), then by receiving 3 spikes from neuron σ_{state}, rule $a^3(a^8)^+/a^{11} \to a^3$ in neuron σ_1 is applied sending 3 spikes to neuron σ_d. The number of spikes in neuron σ_1 becomes $8n+3-11 = 8(n-1)$, simulating the number stored in register 1 is decreased by one. By receiving 3 spikes from neuron σ_1, neuron σ_d accumulates 5 spikes. Neuron σ_d fires by rule $a^5 \to a^4$, sending 4 spikes to each of neurons σ_{s_1} and σ_{s_2}. At that step, system Π switches to use rules from component 2. Neurons σ_{s_1} and σ_{s_2} fire by rule $a^{T+4}/a^{T+3} \to a^{T+1}$, sending $T+1$ spikes to each other as well as sending $T+1$ spikes to neuron σ_{state}. The numbers of spikes in neurons σ_{s_1} and σ_{s_2} become $T+2$. With $T+2$ spikes in neurons σ_{s_1} and σ_{s_2}, rule $a^{T+2} \to a^T$ from component 2 is applied, sending T spikes to each of neurons σ_{state}, σ_{s_1} and σ_{s_2}. The numbers of spikes in neurons σ_{s_1} and σ_{s_2} return to T, and neuron σ_{state} accumulates $2T+i$ spikes. In the next step, component 1 is active, and neuron σ_{state} fires by rule $a^{2T+i}/a^{T+i-k} \to a^4$. Neuron σ_{state} ends with $(2T+i)-(T+i-k) = T+k$ spikes, which simulates system Π starts to simulate instruction l_k of M_u.

The simulation of the SUB instruction $l_i : (\texttt{SUB}(1), l_j, l_k)$ is correct: starting from the simulation of the instruction l_i (neuron σ_{state} has $T+i$ spikes), the system passes to simulate the instruction l_j (with neuron σ_{state} having $T+j$ spikes) if the number stored in register 1 is $n > 0$ and decreased by one, or to simulate the instruction l_k (with neuron σ_{state} having $T+j$ spikes) if the number stored in register 1 is zero.

Similarly, we can check that system Π correctly simulates a SUB instruction $l_i : (\texttt{SUB}(2), l_j, l_k)$ acting one register 2. We omit the details here.

Outputting the computation result

Assume now that the computation in M_u halts, which means that the halt instruction l_m is reached. In system Π, neuron σ_{state} contains $T+m$ spikes and neuron

σ_0 has $7n$ spikes, for n being the number stored in register 0 of M_u. Neuron σ_{state} fires by the rule $a^{T+m} \to a^6$, sending 6 spikes to each of neurons σ_0, σ_1 and σ_2. Neurons σ_1 and σ_2 keep inactive, since no rule can be used. Neuron σ_0 fires by rule $a^{13}(a^7)^+/a^7 \to a^7$, sending 7 spikes to neurons σ_d and σ_{out} in each step. When neuron σ_0 contains 13 spikes, rule $a^{13} \to a^7$ is applied sending 7 spikes to neurons σ_d and σ_{out}, and neuron σ_0 stops firing. Neuron σ_d removes the 7 spikes (received from neuron σ_0) by forgetting rule one step later; but neuron σ_{out} sends one spike into the environment by consuming the 7 spikes one step later. In this way, neuron σ_{out} sends n spikes into the environment, which is exactly the number stored in register 0 when register machine M_u halts.

From the above description of the work of system Π, it is clear that the register machine M_u is correctly simulated by the system Π, *i.e.*, $N(M_u) = N_{tot}(\Pi)$. We can check that the system Π has 8 neurons and neurons in system Π have two components, where extended spiking rules and forgetting rules are used. Therefore, we can conclude $N_{tot}SNP_8CR_2(extend, forg) = NRE$. □

4. Conclusion and Discussions

In this work, we investigated how many neurons are needed to construct Turing universal SN P systems with cooperating rules working in terminating mode. We proved that 8 neurons are enough to construct Turing universal SN P systems with cooperating rules working in terminating mode.

Except for the number of neurons, the number of rules in neurons is also a kind of computation resource. So, it is necessary and of interest to consider the complexity of rules, when we construct "small" universal SN P systems with cooperating rules.

It is of interest to give a boundary between university and non-university of SN P systems with cooperating rules in terms of the number of neurons. Of course, here, each neuron should have a finite set of rules, instead of an infinite set of rules as in [9].

In [7], five cooperating strategies were mentioned. The commotional power of SN P systems with cooperating rules working in other cooperating modes is deserved to be investigated.

In the proof of Theorem 1, the strategy used to simulate register machines is different from the one proposed by A. Păun and G. Păun in [19]. Instead of using a module (a group of neurons) to simulate an instruction of a register machine, we use a class of rules in the neuron labelled with *state* to simulate an instruction of a register machine. In this way, the number of neurons is significantly reduced for constructing Turing universal SN P systems, compared with the results in [19]. The universal SN P systems constructed in [19] are used as function computing devices. It remains open whether we can modify the strategy used in this work to construct small universal SN P systems with cooperating rules working as function computing devices.

Reversible computing devices play vital roles in quantum computing. Reversible SN P systems have been considered in [29], and it was proved that there exist Turing universal reversible SN P systems as number generating and accepting devices. For

further research, it is of interest to construct Turing universal reversible SN P systems with cooperating rules as number generating, accepting and function computing devices.

Acknowledgements. This work was supported by National Natural Science Foundation of China (61033003, 91130034, 61320106005 and 61402187), China Postdoctoral Science Foundation funded project (2014M550389), Ph.D. Programs Foundation of Ministry of Education of China (2012014213008), and Natural Science Foundation of Hubei Province (2011CDA027).

References

[1] CHEN H., FREUND R., IONESCU M., PĂUN GH., PÉREZ-JIMÉNEZ M.J., *On string languages generated by spiking neural P systems*, Fundamenta Informaticae, 2007, Vol. **75(1)**, pp. 141–162.

[2] CHEN H., IONESCU M., ISHDORJ T.O., PĂUN A., PĂUN GH., PÉREZ-JIMÉNEZ M.J., *Spiking neural P systems with extended rules, universality and languages*, Natural Computing, 2008, Vol. **7(2)**, pp. 147–166.

[3] CSUHAJ-VARJU E., DASSOW J., KELEMEN J., PĂUN GH., *Grammar systems. a grammatical approach to distribution and cooperation*, Gordon and Breach, London, 1994.

[4] IONESCU M., PĂUN GH., YOKOMORI T., *Spiking neural P systems*, Fundamenta Informaticae, 2006, Vol. **71(2)**, pp. 279–308.

[5] KOREC I., *Small universal register machines*, Theoretical Computer Science, 1996, Vol. **168(2)**, pp. 267–301.

[6] METTA V.P., KELEMENOVÁ A., *Universality of spiking neural P systems with anti-spikes*, Theory and Applications of Models of Computation, Springer, 2014, pp. 352–365.

[7] METTA V.P., RAGHURAMAN S., KRITHIVASAN K., *Spiking neural P systems with cooperating rules*, Proceedings of the 15th Conference on Membrane Computing, August 20–22, 2014, Prague, Czech Republic, 267–282.

[8] Minsky M.L., *Computation: finite and infinite machines*, Prentice Hall, 1967.

[9] NEARY T., *A boundary between universality and non-universality in extended spiking neural P systems*, Language and Automata Theory and Applications, Springer, 2010, pp. 475–487.

[10] NEARY T., *A universal spiking neural P system with 11 neurons*, Proceedings of the 11th International Conference on Membrane Computing, Springer, 2010.

[11] OLLINGER N., *The quest for small universal cellular automata*, Proceedings of Automata, Languages and Programming, Springer, 2002, pp. 318–329.

[12] PAN L., PĂUN GH., PÉREZ-JIMÉNEZ M.J., *Spiking neural P systems with neuron division and budding*, Science China Information Sciences, 2011, Vol. **54(8)**, 2013, pp. 1596–1607.

[13] PAN L., PĂUN GH., *Spiking neural P systems with anti-spikes*, International Journal of Computers, Communications & Control, 2009, Vol. **4(3)**, pp. 272–283.

[14] PAN L., WANG J., HOOGEBOOM H.J., *Spiking neural P systems with astrocytes*, Neural Computation, 2012, Vol. **24(3)**, pp. 805–825.

[15] PAN L., ZENG X., *A note on small universal spiking neural P systems*, Lecture Notes in Computer Science, 2010, Vol. **5957**, pp. 436–447.

[16] PAN L., ZENG X., *Small universal spiking neural P systems working in exhaustive mode*, IEEE Transactions on NanoBioscience, 2010, Vol. **10(2)**, pp. 99–105.

[17] PAN L., ZENG X., ZHANG X., *Small universal asynchronous spiking neural P systems*, Proceedings of IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications, 2010, pp. 622–630.

[18] PĂUN GH., *Membrane computing: an introduction*, Springer-Verlan, 2002.

[19] PĂUN A., PĂUN GH., *Small universal spiking neural P systems*, BioSystems, 2007, Vol. **90**(1), pp. 48–60.

[20] PĂUN A., SIDOROFF M., *Sequentiality induced by spike number in SNP systems: small universal machines*, Lecture Notes in Computer Science, 2012, Vol. **7184**, pp. 333–345.

[21] PĂUN GH., *Spiking neural P systems with astrocyte-like control*, Journal of Universal Computer Science, 2007, Vol. **13(11)**, pp. 1707–1721.

[22] PĂUN, GH., PÉREZ-JIMÉNEZ, M.J., *Spiking neural P systems: an overview*, In: A.B. Porto, A. Pazos, W.B. (eds.) Advancing Artificial Intelligence through Biological Process Applications, 2008, pp. 60–73. PA: Medical Information Science Reference, Hershey.

[23] ROGOZHIN Y., *Small universal turing machines*, Theoretical Computer Science, 1996, Vol. **168(2)**, pp. 215–240.

[24] PĂUN GH., ROZENBERG G., SALOMAA A. (eds.), *Handbook of Membrane Computing*, Oxford University Press, Cambridge, 2010.

[25] SONG T., JIANG Y., SHI X., ZENG X., *Small universal spiking neural P systems with anti-spikes*, Journal of Computational and Theoretical Nanoscience, 2013, Vol. **10(4)**, pp. 999–1006.

[26] SONG T., PAN L., PĂUN GH., *Asynchronous spiking neural P systems with local synchronization*, Information Sciences, 2013, Vol. **219**, pp. 197–207.

[27] SONG T., PAN L., PĂUN, GH., *Spiking neural P systems with rules on synapses*, Theoretical Computer Science, 2014, Vol. **529(10)**, pp. 82–95.

[28] SONG T., PAN L., WANG J., VENKAT I., SUBRAMANIAN K., ABDULLAH R., *Normal forms of spiking neural P systems with anti-spikes*, IEEE Transactions on NanoBioscience, Vol. **11(4)**, pp. 352–359.

[29] SONG T., SHI X., XU J., *Reversible spiking neural P systems*, Frontiers of Computer Science, 2013, Vol. **7(3)**, pp. 350–358.

[30] SONG T., WANG X., *Homogenous spiking neural P systems with inhibitory synapses*, Neural Processing Letters, 2014, DOI: 10.1007/s11063-014-9352-y.

[31] SONG T., WANG X., ZHANG Z., CHEN Z., *Homogenous spiking neural P systems with anti-spikes*, Neural Computing and Applications, 2014, Vol. **24**, pp. 1833–1841.

[32] ZENG X., ZHANG X., PAN L., *Homogeneous spiking neural P systems*, Fundamenta Informaticae, 2009, Vol. **97**, pp. 1–20.

[33] ZHANG X., ZENG X., PAN L., *Smaller universal spiking neural P systems*, Fundamenta Informaticae, 2008, Vol. **87**(1), pp. 117–136.