Mathematical and Computer Modelling 52 (2010) 1940–1946

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

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

A weakly universal spiking neural P system

Xiangxiang Zeng, Chun Lu, Linqiang Pan *

Key Laboratory of Image Processing and Intelligent Control, Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, PR China

ARTICLE INFO

Article history:
Received 23 September 2009
Received in revised form 25 December 2009
Accepted 31 January 2010

Keywords:
Membrane computing
Spiking neural P system
Universality
Register machine

ABSTRACT

Looking for small universal computing devices is a natural and well investigated topic in computer science. Recently, this topic was also investigated in the framework of spiking neural P systems. One of the small universality results is that a small weakly universal extended spiking neural P system with 12 neurons was constructed. In this paper, a new way is introduced for simulating register machines by spiking neural P systems, where only one neuron is used for all instructions of the register machine; in this way, we can use less neurons to construct universal spiking neural P system. Specifically, we give a smaller weakly universal spiking neural P system that uses extended rules and has only 9 neurons.

1. Introduction

Spiking neural P systems (in short, SN P systems) are a class of computing devices introduced in [1], which is an attempt to incorporate the idea of spiking neurons into the area of membrane computing. Although the research of SN P systems was initiated recently, in the year 2006, it became a hot research area and there have been a lot of papers on this topic. We refer to the respective chapter of [2] for general information in this area, and to the membrane computing web site from [3] for details.

Informally, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph, called the *synapse graph*. The content of each neuron consists of a number of copies of a single object type, called the *spike*. The spikes evolve by means of *standard spiking rules*, which are of the form $E/a^c \rightarrow a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1$, $d \geq 0$. The meaning is that 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. There also are *forgetting rules*, of the form $a^s \rightarrow \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes. *Extended rules* were considered in [4]: these rules are of the form $E/a^c \rightarrow a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain a generalization of both standard spiking and forgetting rules. The system works in a synchronized manner, i.e., in each time unit, each neuron which can apply a rule should do it. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system — it might be infinite if the computation does not stop.

In [1], SN P systems were proved to be computationally complete. An interesting problem is that how many neurons are necessarily used to construct a universal SN P system. Of course, we hope that the number of neurons used in universal SN P system is as small as possible. In this paper, we consider a small universal SN P system in the case of weak universality. The formal definitions of weak universality and strong universality will be given in Section 2. Here, we only mention that

* Corresponding author. Tel.: +86 27 87556070, +86 27 87543563; fax: +86 27 87543130.
E-mail addresses: xzeng@foxmail.com (X. Zeng), luchun.et@gmail.com (C. Lu), lqpan@mail.hust.edu.cn (L. Pan).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.04.017

strong universality has strict conditions regarding the encoding of input and decoding of output. Weak universality has more relaxed conditions regarding the encoding of input and decoding of output.

Small strong universality was firstly considered in SN P systems by Andrei Păun and Gheorghe Păun in [5], where strongly universal SN P systems were obtained with 84 neurons using standard rules and with 49 neurons using extended rules. Then an improvement was obtained in [6], the number of neurons was reduced to 68 for standard rules (or 43 for extended rules). On the other hand, small weak universality was firstly considered by Turlough Neary in [7], where a small weakly universal SN P system is constructed with 12 neurons using extended rules.

The proofs of the above results are based on simulating register machines. In particular, the proof of the result in [7] starts from a weakly universal register machine with only 2 registers. In this paper, by simulating the same register machine, a smaller SN P system with only 9 neurons is constructed, where 3 neurons are used to take care of inputting spikes from the environment; 2 neurons are associated with 2 registers; one neuron is used to output the result of computation, one neuron is used for all instructions of the register machine; 2 auxiliary neurons are used between the neuron associated with instructions and neurons associated with registers, which works as a "sieve" (only spikes that we want to pass through these neurons can pass these neurons).

The paper is structured as follows. In the next section, we introduce some necessary prerequisites. Section 3 introduces spiking neural P systems. In Section 4 we prove that an extended SN P system (without delay) with only 9 neurons is weakly universal. Conclusions and remarks are given in Section 5.

2. Prerequisites

We assume the reader to be familiar with (basic elements of) language theory [8], as well as basic membrane computing [9] (for more updated information about membrane computing, please refer to [3]), hence we directly introduce here some notations and basic definitions.

For an alphabet V, let V^* denotes the set of all finite strings over V, with the empty string denoted by λ. The set of all non-empty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

A regular expression over an alphabet 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 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 are omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

A register machine is a construct $M = (z, H, l_1, l_h, I)$, where z is the number of registers, H is the set of instruction labels, l_1 is the start label, 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 register r and then go to one of the instructions with labels l_j or l_k, non-deterministically chosen),
- 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 M generates a set $N(M)$ of numbers in the following way: we start with all registers being empty (i.e., storing the number zero), we apply the initial instruction with label l_1 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction l_h, then the number n present in specified register r_s at that time is said to be generated by M. If the computation does not halt, then no number is generated. It is known (see, e.g., [10]) that register machines generate all sets of numbers which are Turing computable.

A register machine can also compute any Turing computable function: we introduce the arguments in specified registers $r_1, \ldots, r_k$ (without loss of the generality, we may assume that we use the first k registers), we start with the initial instruction with label l_1, and if the register machine stops (with the instruction with label l_h), then the value of the function is placed in another specified register r_t, with all registers different from r_t being empty. The partial function computed in this way is denoted by $M(n_1, n_2, \ldots, n_k)$. In the computing form, the register machines can be considered deterministic, without losing the Turing completeness; then, the ADD instructions l_i : (ADD(r), l_j, l_k) have $l_j = l_k$ (and the instruction is written in the form l_i : (ADD(r), l_j)).

In [11], there are two different notions of universality defined as follows—strong universality and weak universality.

Definition 1. Let $(\phi_0, \phi_1, \phi_2, \ldots)$ be a fixed admissible enumeration of the set of unary partial recursive functions.

(i) A register machine M will be called strongly universal if there is a recursive function g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = M(g(x), y)$.
(ii) A register machine M will be called weakly universal if there are recursive functions f, g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = f(M(g(x, y)))$.

In this paper, we work with weak universality and we use deterministic register machines. The following theorem about weakly universal register machines is given in Minsky's book [10].

Theorem 1. *There exist weakly universal deterministic register machines that use only two registers.*

For the details of how to find such a register machine, we refer to Section 14.1 of book [10], where Minsky introduced a procedure to construct this kind of register machines. Here we mention that this kind of register machines uses deterministic ADD and SUB instructions, and has two registers (r_1 and r_2), with the input introduced in register r_1 of the universal machine, and the result obtained also in register r_1.

Convention: when evaluating or comparing the power of two number generating/accepting devices, number zero is ignored.

3. Spiking neural P systems

We briefly recall the basic notions concerning SN P systems. For more details on these systems, please refer to [1].

A *spiking neural P system* of degree $m \geq 1$ is a construct of the form

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

where:

1. $O = \{a\}$ is the singleton alphabet (a is called spike);
2. $\sigma_1, \ldots, \sigma_m$ are neurons, of the form
$$\sigma_i = (n_i, R_i), \quad 1 \leq i \leq m,$$
where:
   (a) $n_i \geq 0$ is the initial number of spikes contained in σ_i;
   (b) R_i is a finite set of rules of the following two forms:
      (1) $E/a^c \to a^p; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$, $p \geq 1$, with the restriction $c \geq p$;
      (2) $a^s \to \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \to a^p; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (synapses between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicates the input and the output neurons, respectively.

If we always have $p = 1$ for all rules of the form $E/a^c \to a^p; d$, then the rules are said to be of the standard type, else they are called by extended rules.

The rules of type (1) are firing (we also say spiking) rules, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E)$, $k \geq c$, then the rule $E/a^c \to a^p; d \in R_i$ can be applied. This means consuming (removing) c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces p spikes after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then these spikes are emitted immediately, if $d = 1$, then these spikes are emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t + 1, \ldots, t + d - 1$ 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). In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules).

The rules of type (2) are forgetting rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \to a; d$ has $E = a^c$, then we will write it in the simplified form $a^c \to a; d$.

If a rule $E/a^c \to a; d$ has $d = 0$, then we will write it in the simplified form $E/a^c \to a$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. Since two firing rules, $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case, only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are used in the sequential manner in each neuron, at most one in each step, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the total number of spikes contained in the neuron.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron, with all neurons being open. During the computation, a configuration of the system is described by both the number of spikes present in each neuron and by the state of the neuron, more precisely, by the number of steps to count down until it becomes open (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, $i = 1, 2, \ldots, m$; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

In next section, as usual, an SN P system is represented graphically, which may be easier to understand than in a symbolic way. We give an oval with rules inside to represent a neuron, and directed graph to represent the structure of SN P system: the neurons are placed in the nodes of a directed graph and the directed edges represent the synapses; the input neuron has an incoming arrow and the output neuron has an outgoing arrow, suggesting their communication with the environment.

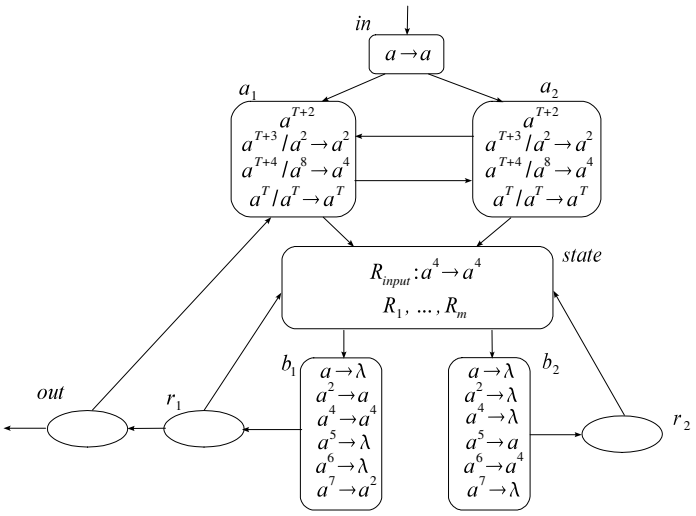

Fig. 1. SN P system Π.

4. A small weakly universal SN P system

In this section, we give our small weakly universal SN P system (where extended rules, producing more than one spikes at a time, are used) by simulating a weakly universal register machine M that has only two registers.

Let $M = (2, H, l_1, l_m, I)$ be a weakly universal register machine, where $I = \{l_1, l_2, \ldots, l_m\}$, r_1 is the input register and the output register, l_1 is the initial instruction and l_m is the halt instruction, respectively. In what follows, we construct a specific SN P system Π to simulate M.

An SN P system Π is given in Fig. 1, where neurons σ_{in}, σ_{a_1}, σ_{a_2} are used to load spikes to neurons σ_{state}, σ_{r_1}; neuron σ_{state} is associated with all instructions of M; neurons σ_{r_1} and σ_{r_2} are associated with registers r_1 and r_2; neurons σ_{b_1} and σ_{b_2} work like a "sieve" (in the sense that only spikes that we want to move from neuron σ_{state} to neuron associated with register r can pass neurons σ_{b_1} and σ_{b_2}, and reach neuron associated with register r); neuron σ_{out} is used to output the result of computation.

In system Π, each neuron is assigned with a set of rules, see Table 1, where $T = 4(m + 1) + 3$, $P(i) = 4(i + 1)$, $i = 1, 2, \ldots, m$. In neuron σ_{state}, there are $m + 1$ groups of rules R_{input}, R_1, R_2, ..., R_m, where: the set of rules R_{input} takes care of loading spikes into σ_{r_1}; for each ADD instruction l_i : $(\mathtt{ADD}(\mathtt{r}), l_j)$, the set of rules $R_i = \{a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{add(r)}\}$ is associated, where $add(1) = 4$, $add(2) = 6$; for each SUB instruction l_i : $(\mathtt{SUB}(r), l_j, l_k)$, the set of rules $R_i = \{a^{P(i)}(a^T)^+/a^{T+3} \to a^{sub(r)}, a^{P(i)-1}(a^T)^+/a^{P(i)-1+T-P(j)} \to a, a^{P(i)-2}(a^T)^+/a^{P(i)-2+T-P(k)} \to a\}$ is associated, where $sub(1) = 2, sub(2) = 5$; for instruction l_m : HALT, $R_m = \{a^{P(m)}(a^T)^+/a^{P(m)} \to a^7\}$. If the number of spikes in neuron σ_{state} is of the form $P(i) + sT$ for some $s \geq 1$ (that is, if the number of spikes is n, then $n \equiv P(i) \pmod{T}$; the value of multiplicity of T does not matter with the restriction that it should be greater than 0), then system Π starts to simulate instruction l_i (in particular, having the number of spikes of the form $P(1) + sT = 4(1 + 1) + sT = 8 + sT$ in σ_{state}, system Π starts to simulate the initial instruction l_1 of M; with $P(m) + sT = 4(m + 1) + sT$ spikes, starts to output the result of computation). That is why we use the label *state* for this neuron, and the function of this neuron is somewhat similar with "control unit" in Turing machine.

Initially, all neurons have no spike, with exception that each of neurons $\sigma_{a_1}, \sigma_{a_2}$ contains $T + 2$ spikes. During the computation of M, if the register r hold the number $n \geq 0$, then the associated neuron σ_r will contains $4n$ spikes. In what follows, we check the simulation of register machine M by system Π.

Reading from environment and inputting to neuron σ_{r_1}. The input to Π is read into the system via the input neuron σ_{in} as shown in Fig. 1. If the input to M is x then the binary sequence $z = 10^{x-1}1$ is read in by the input neuron σ_{in}. This means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z and no spike otherwise.

The input part works as follows. After receiving the first spike, neuron σ_{in} fires immediately. At step 3, neurons σ_{a_1} and σ_{a_2} become active, sending two spikes to each other and spiking again in the next step; at the same time, 4 spikes from them arrive in neuron σ_{state}. This process lasts until step $x + 2$.

From step 3 to step $x + 2$, neuron σ_{state} receives four spikes at every step, sending 4 spikes to neurons σ_{b_1} and σ_{b_2} by the rule $a^4 \to a^4$. With 4 spikes inside, neuron σ_{b_1} spikes, sending 4 spikes to neuron σ_{r_1}, which corresponds to increase the number stored in register r_1 by one; With 4 spikes inside, neuron σ_{b_2} forgets. Thus, at every step, the content of neuron σ_{r_1} increases by 4. This process lasts for x steps, and in this way neuron σ_{r_1} is loaded with $4x$ spikes.

At step $x + 1$, another spike from environment arrives in the system, neuron σ_{in} spikes again at the next step. After receiving the spike, neurons σ_{a_1} and σ_{a_2} have $T + 4$ spikes inside, each of them sends four spikes to neuron σ_{state} by the rule

Table 1
The rules associated with neurons in system Π.

Neurons	Associated rules
σ_{in}	$a \to a$
$\sigma_{a_1}, \sigma_{a_2}$	$a^{T+3}/a^2 \to a^2, a^{T+4}/a^8 \to a^4, a^T \to a^T$
σ_{b_1}	$a \to \lambda, a^2 \to a, a^4 \to a^4, a^5 \to \lambda, a^6 \to \lambda, a^7 \to a^2$
σ_{b_2}	$a \to \lambda, a^2 \to \lambda, a^4 \to \lambda, a^5 \to a, a^6 \to a^4, a^7 \to \lambda$
σ_{r_1}	$a \to a, a^2 \to a, a(a^4)^+/a^5 \to a^2, a^2(a^4)^+/a^4 \to a^4$
σ_{r_2}	$a \to a, a(a^4)^+/a^5 \to a^2$
σ_{out}	$a \to a, a^3 \to \lambda, a^5 \to a$
σ_{state}	$R_{state} = R_{input} \cup R_0 \cup R_1 \cup \cdots \cup R_m$, where: $R_{input} = \{a^4 \to a^4\}$; $R_i = \{a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{add(r)}\}$, if $l_i : (\mathtt{ADD}(r), l_j) \in I$; $R_i = \{a^{P(i)}(a^T)^+/a^{T+3} \to a^{sub(r)}, a^{P(i)-1}(a^T)^+/a^{P(i)-1+T-P(j)} \to a,$ $a^{P(i)-2}(a^T)^+/a^{P(i)-2+T-P(k)} \to a\}$, if $l_i : (\mathtt{SUB}(r), l_j, l_k) \in I$; $R_m = \{a^{P(m)}(a^T)^+/a^{P(m)} \to a^7\}$, for instruction l_m : HALT

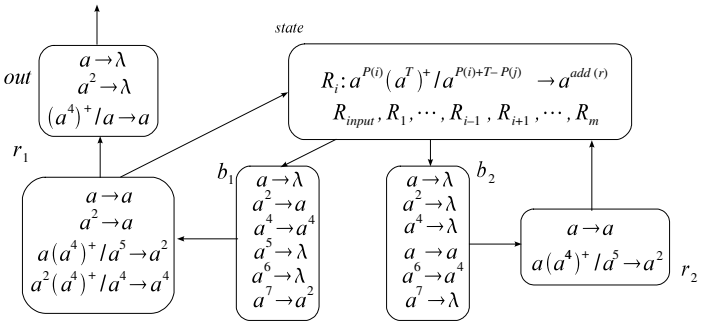

Fig. 2. Simulating $l_i : (\mathtt{ADD}(r), l_j)$.

$a^{T+4}/a^8 \to a^4$. From step $x+3$ on, with T spikes inside, neurons σ_{a_1} and σ_{a_2} send T spikes to each other at every step and spike again in the next step, rule $a^T \to a^T$ is continuously applied.

With $8 + sT = P(0) + sT$ (for some $s \geq 1$) spikes in neuron σ_{state} at step $x+3$, the system is triggered to simulate the initial instruction l_1 of M. In general, the simulation of an ADD or SUB instruction with label l_i starts by introducing $P(i) + sT$ (for some $s \geq 1$) spikes in neuron σ_{state}.

Simulating an ADD instruction. The part of system for simulating a deterministic ADD instruction: $l_i : (\mathtt{ADD}(r), l_j)$ is depicted in Fig. 2. Recall that the number of spikes $add(r)$ in the rule $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{add(r)}$ is used for indicating the ADD operation to register r, which is defined as $add(1) = 4$, $add(2) = 6$.

Assume that we are at a step when we have to simulate an ADD instruction $l_i : (\mathtt{ADD}(1), l_j)$, with the number of spikes of the form $P(i) + sT$ (for some $s \geq 1$) in neuron σ_{state}. The rule $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{add(1)}$ is applied by neuron σ_{state} and emits $add(1) = 4$ spikes, at the next step, neuron σ_{a_1} sends 4 spikes to neuron σ_{r_1}, which corresponds to adding register r_1 of M with 1. In neuron σ_{a_2}, these 4 spikes will be forgotten by rule $a^4 \to \lambda$.

After consuming $P(i) + T - P(j)$ spikes by the rule $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{add(1)}$, the number of spikes in neuron σ_{state} is of the form $P(j) + sT$ (for some $s \geq 1$), hence we pass to the next instruction l_j.

Similarly, the simulation of ADD instruction $l_i : (\mathtt{ADD}(2), l_j)$ acting on register 2 can be checked, which we omit here.

The simulation of the ADD instruction is correct: we have increased the number of spikes in neuron σ_{r_1} or σ_{r_2} by four, and we have passed to the simulation of one of the instructions l_j and l_k non-deterministically.

Simulating a SUB instruction. The part of system for simulating a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is shown in Fig. 3, where $sub(r)$ in the rule $a^{P(i)}(a^T)^+/a^{T+3} \to a^{sub(r)}$ is used for indicating the SUB operation to register r, $sub(1) = 2$, $sub(2) = 5$.

Let us assume $r = 1$ and neuron σ_{state} emits $sub(1) = 2$ spikes at time t (for SUB instruction acting on register 2, the simulation can be checked similarly). At time $t+1$, neuron σ_{b_2} forgets these two spikes, only neuron σ_{b_1} fires and sends one spike to neuron σ_{r_1}. For neuron σ_{r_1}, there are the following two cases.

(1) The number of spikes in neuron σ_{r_1} at time t is $4n$ with $n > 0$. Then at step $t+2$, the rule $a(a^4)^+/a^5 \to a^2$ consumes 4 spikes, sending 2 spikes to neuron σ_{state}. After receiving these 2 spikes, the number of spikes in neuron σ_{state} is of the form $P(i) - 1 + sT$ (for some $s \geq 1$), so the rule $a^{P(i)-1}(a^T)^+/a^{P(i)+T-1-P(j)} \to a$ can be applied. Consuming $P(i) + T - 1 - P(j)$ at time $t+3$ by the rule $a^{P(i)-1}(a^T)^+/a^{P(i)+T-1-P(j)} \to a$, there are $P(j) + sT$ spikes leaving in neuron σ_{state}, which means that the next execution instruction will be l_j. Note that the one spike emitted by neuron σ_{state} will immediately be forgotten by neurons σ_{a_1} and σ_{a_2} at the next step because of the rule $a \to \lambda$ in them.

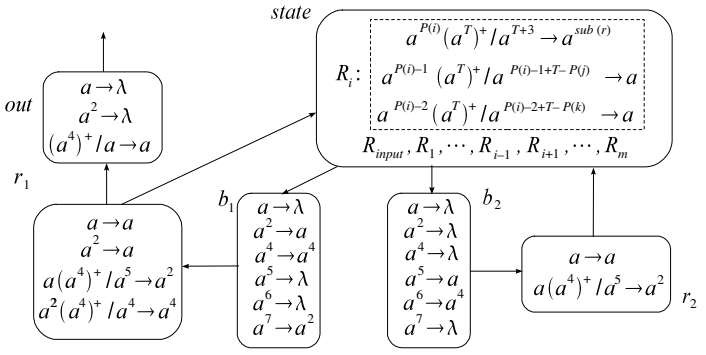

Fig. 3. Simulating l_i : (SUB(r), l_j, l_k).

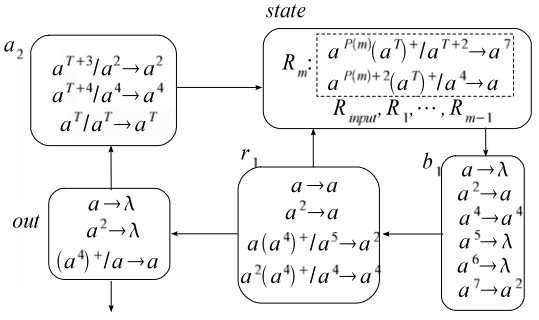

Fig. 4. Outputting the result and halting.

(2) The number of spikes in neuron σ_{r_1} at time t is 0. Then the rule $a \to a$ consumes the single spike present in the neuron and sends 1 spike to neuron σ_{state} at time $t+2$. In this case, neuron σ_{state} contains $P(i)-2+sT$ spikes at time $t+3$, the rule $a^{P(i)-2}(a^T)^+/a^{P(i)+T-2-P(k)} \to a$ will consume $P(i)+T-2-P(k)$ spikes, leaving $P(k)+sT$ spikes in neuron σ_{state}, so system Π starts to simulate the instruction l_k.

The simulation of the SUB instruction is correct: starting from the simulation of instruction l_i, we passed to simulate the instruction l_j if the register was non-empty and decreased by one, and to simulate instruction l_k if the register is empty.

Remark. In the set of rules R_i associated with a SUB instruction l_i, the regular expressions have numbers $P(i)$, $P(i)-1$, $P(i)-2$, $P(i)-3$. Because $P(i) = 4(i+1)$ for each instruction l_i, which implies that $\{P(i_1), P(i_1)-1, P(i_1)-2, P(i_1)-3\} \cap \{P(i_2), P(i_2)-1, P(i_2)-2, P(i_2)-3\} = \emptyset$, for $i_1 \neq i_2$, the simulation of SUB instructions do not interfere with each other. On the other hand, in the set of rules R_i associated with an ADD instruction l_i, the regular expressions have number $P(i)$, it is not difficult to see that the simulations of an ADD instruction and a SUB instruction do not interfere with each other too. That is why we take $P(i)$ as a multiplicity of number 4.

Outputting the result and halting. Having the result of the computation in register r_1, we can output the result by means of the part of system from Fig. 4.

Assume now that the computation in M halts, which means that the halt instruction l_m : HALT is reached. For system Π, this means that neuron σ_{state} contains $P(h)+sT$ (for some $s \geq 1$) spikes, neuron σ_{r_1} stores $4M(x)$ spikes ($M(x)$ is the result computed by M). Having $P(h)+sT$ spikes inside, neuron σ_{state} gets fired and emits 7 spikes. In the next step, neuron σ_{b_2} forgets these 7 spikes from σ_{state}, neuron σ_{b_1} fires and sends 2 spikes to neuron σ_{r_1}. In this way, neuron σ_{r_1} has $4M(x)+2$ spikes, hence its rule $a^2(a^4)^+/a^4 \to a^4$ can be applied and sends 4 spikes to neuron σ_{state} and σ_{out}. Four spikes arrive in neuron σ_{state}, which consumes these spikes immediately by rule $a^{P(m)+2}(a^T)^+/a^4 \to a$ (note that the one spike emitted will be forgotten by neuron σ_{b_1} and σ_{b_2} at the next step); 4 spikes arrive in the output neuron σ_{out}, which spikes for the first time, consuming one spike (after receiving the first spike from the output neuron, neuron σ_{a_2} has $2T+1$ spikes inside and is "blocked"). From the next step on until exhausting the spikes from neuron σ_{r_1}, the output neuron receives 4 spikes at each step; the number of spikes in neuron σ_{out} is of the form $4k+3$ for some $k \geq 0$, and neuron σ_{out} cannot use its rules. When neuron σ_{r_1} has only two spikes, its rule $a^2 \to a$ can be applied and sends 1 spike to the output neuron, thus the output neuron spikes for the second (and last) time. From then on, there is no rule can be applied in system Π, and in this way it halts.

From the above description, it is clear that Π can simulate each computation of the weakly universal register machine M, so Π is also weakly universal. It is easy to observe from Fig. 1 that the universal system Π has 9 neurons in total. We formulate this result as a theorem.

Theorem 2. *There is a weakly universal SN P system with extended rules (without delay) having* 9 *neurons.*

5. Conclusions and remarks

In this paper, we have obtained an improvement (from 12 to 9) of the number of neurons for small weakly universal SN P system. A weakly universal SN P system with 9 neurons looks already quite small. However, it remains open whether this system is optimal; that is how to prove this system is smallest in the sense that we cannot construct a universal SN P system with less neurons.

In our small SN P system, 3 neurons are used to take care of inputting spikes from the environment; 2 neurons are associated with 2 registers; one neuron is used to output the result of computation, one neuron is used for all instructions of the register machine; 2 auxiliary neurons are used between the neuron associated with instructions and neurons associated with registers, which works as a "sieve" (only spikes that we want to pass through these neurons can pass these neurons). Can we remove these 4 auxiliary neurons to get smaller universal SN P systems? One possible way of removing these auxiliary neurons is to use more rules in the neuron associated with instructions realizing the function of "sieve".

The reachability question for spiking neural P systems is as follows: given a configuration c_x of a spiking neural P system does it ever enter a configuration c_y. It is worth noting that using the results in Theorem 2 smaller spiking neural P systems with undecidable reachability questions may be given. Such systems may be given by removing the output neuron and the 3 neurons for initializing the system from Π. Thus, there exists spiking neural P system with 5 neurons which has undecidable reachability question.

Strong universality has been considered in several variants of systems such as standard SN P systems, extended SN P systems, asynchronous SN P systems, SN P systems with exhaustive use of rules (general information about these systems can be found, e.g., in [12–14]). Small weak universality in these variants of systems also deserves to be investigated, checking whether there are some unexpected results.

Acknowledgements

The work was supported by National Natural Science Foundation of China (Grant Nos. 30670540, 60674106, 30870826, 60703047, and 60533010), 863 Program of China (2009AA012413), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180).

References

[1] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundamenta Informaticae 71 (2–3) (2006) 279–308.
[2] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, 2010.
[3] The P System Web Page: http://ppage.psystems.eu.
[4] H.M. Chen, M. Ionescu, T.-O. Ishdorj, M.J. Pérez-Jiménez, Spiking neural P systems with extended rules, in: Proc. Fourth Brainstorming Week on Membrane Computing, Sevilla, Spain, 2006, pp. 241–266.
[5] A. Păun, Gh. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 48–60.
[6] X. Zhang, X. Zeng, L. Pan, Smaller universal spiking neural P systems, Fundamenta Informaticae 87 (1) (2008) 117–136.
[7] T. Neary, A Small universal spiking neural P system, in: Proc. 7th International Workshop on Computing with Biomolecules, Wien, Austria, 2008, pp. 65–74.
[8] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 3, Springer-Verlag, Berlin, 1997.
[9] Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berlin, 2002.
[10] M. Minsky, Computation — Finite and Infinite Machines, Prentice Hall, New Jersey, 1967.
[11] I. Korec, Small universal register machines, Theoretical Computer Science 168 (2) (1996) 267–301.
[12] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems with exhaustive use of rules, International Journal of Unconventional Computing 3 (2007) 135–154.
[13] M. Cavaliere, O. Egecioglu, O.H. Ibarra, S. Woodworth, M. Ionescu, Gh. Păun, Asynchronous spiking neural P systems, Theoretical Computer Science 410 (24–25) (2009) 2352–2364.
[14] X. Zhang, Y. Jiang, L. Pan, Small universal spiking neural P systems with exhaustive use of rule, in: Proc. Third International Conference on Bio-Inspired Computing: Theory and Application, Adelaide, Austrilia, 2008, pp. 117–128.