A Note on Small Universal Spiking Neural P Systems

Linqiang Pan*, Xiangxiang Zeng

Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
lqpan@mail.hust.edu.cn, xzeng@foxmail.com

Summary. In the "standard" way of simulating register machines by spiking neural P systems (in short, SN P systems), one neuron is associated with each instruction of register machine that we want to simulate. In this note, a new way is introduced for simulating register machines by SN P systems, where only one neuron is used for all instructions of a register machine; in this way, we can use less neurons to construct universal SN P systems. Specifically, a universal system with extended rules (without delay) having 12 neurons is constructed.

1 Introduction

The spiking neural P systems (in short, SN P systems) were introduced in [1], and then investigated in a large number of papers. We refer to the respective chapter of [6] for general information in this area, and to the membrane computing web site from [10] 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 rules assigned to neurons allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes). An output can be defined in the form of the spike train produced by a specified output neuron.

Looking for small universal computing devices of various types is a well investigated issue in computer science, see, e.g. [2, 7], and the references therein. Recently, this issue was considered also in the case of SN P systems [4], where a universal SN P system was obtained using 84 neurons for standard rules and 49 neurons for extended rules in the case of computing functions; used as generators of sets of numbers, a universal system with standard rules (resp. extended rules)

* Corresponding author. Tel.: +86-27-87556070. Fax: +86-27-87543130.

having 76 neurons (resp. 50 neurons) was found. An improvement is presented in [9] in the sense that less neurons are used to construct a universal SN P system. Specifically, in the computing function mode, 68 neurons (resp. 43 neurons) are used to construct a universal SN P system with standard rules (resp. extended rules); in the number generating mode, universal SN P systems are obtained with 64 neurons (resp. 43 neurons) using standard rules (resp. extended rules). All of the above universal SN P systems are obtained by simulating a register machine from [2], where a neuron is associated with each register of the register machine that we want to simulate; a neuron is associated with each instruction of the register machine; some auxiliary neurons are also used. If in the register machine that we want to simulate, there are m instructions and n registers, then the number of neurons in the universal SN P system obtained by this way is not less than $m+n$.

In this note, we present a new approach to a simulate register machine, where one neuron (denoted by σ_{state}) is used for all instructions of the register machine. The function of neuron σ_{state} is similar with "the finite set of states" in a Turing machine. In this way, universal SN P systems with less neurons can be obtained. Specifically, a universal SN P system is constructed with extended rules (without delay) having 12 neurons.

The rest of this paper is organized as follows. In the next section, we introduce some necessary prerequisites. In Section 3, a small universal SN P system is constructed. Conclusions and remarks are presented in Section 4.

2 Prerequisites

We assume the reader to be familiar with (basic elements of) language theory [8], as well as basic membrane computing [5] (for more updated information about membrane computing, please refer to [10]), hence we directly introduce some basic notions and notations including register machines and SN P systems.

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 nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write 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, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

2.1 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:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (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),
- $l_h : \mathtt{HALT}$ (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 instruction with label l_0 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, then the number n present in specified register r_0 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., [3]) that register machines generate all sets of numbers which are Turing computable, even using register machines with only three registers as well as registers 1 and 2 being empty whenever the register machine halts, where we assume that the three registers are labeled with 0, 1, 2.

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

2.2 Spiking Neural P Systems

We briefly recall the basic notions concerning spiking neural P systems (in short, SN P systems). For more details on such kind of systems, please refer to [1].

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

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

1. $O = \{a\}$ is the singleton alphabet (a is called spike);
2. $\sigma_1, \ldots, \sigma_m$ are neurons, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m, \text{ where:}$$

   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) $E/a^c \rightarrow a^p; d$, 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) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ 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 $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. In this note, we use SN P systems as number generating devices, we start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron.

In the 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.

3 A Small Universal SN P System

In this section we shall give a small universal SN P system (where extended rules, producing more than one spikes at a time, are used).

Let $M_u = (3, H, l_0, l_{m-2}, I)$ be a universal register machine with 3 registers labeled by 0, 1, 2, where $m \geq 2$, $H = \{l_0, l_1, l_2, \dots, l_{m-2}\}$ is the set of instruction labels, l_0 is the start label (labeling an ADD instruction) and l_{m-2} is the halt label (assigned to instruction HALT), I is the set of instructions.

We modify the universal register machine M_u such that the register where we place the result is not subject to subtraction operations in the new register machine. To this aim, we add a further register 3 to output the result, and replace the halt instruction l_{m-2} of M_u with the following instructions:

$$l_{m-2} : (\mathtt{SUB}(0), l_{m-1}, l_m), \quad l_{m-1} : (\mathtt{ADD}(3), l_{m-2}), \quad l_m : \mathtt{HALT}.$$

The new register machine M'_u has 4 registers, $m + 1$ instructions (m ADD and SUB instructions, and one halt instruction). In the following proof of Theorem 1, a small universal SN P system is constructed by simulating the register machine M'_u.

Theorem 1. *There is a universal SN P system with extended rules (without delay) having 12 neurons.*

Proof. We shall present an SN P system Π with 12 neurons to simulate register machine M'_u. The structure of system Π is given in Figure 1, where spiking rules are omitted, which will be specified below. In system Π, neuron σ_{state} contains all spiking rules associated with all instructions of M'_u (it is a point different with the "standard" way of simulating register machines by SN P systems, where one neuron is associated with each instruction of register machine that we want to simulate); neurons σ_i and σ_{a_i} ($i = 0, 1, 2, 3$) are associated with registers $0, 1, 2, 3$; neuron σ_{out} is used to output the result of computation; auxiliary neurons σ_{b_1}, σ_{b_2} are used to send a fixed number of spikes to neuron σ_{state} at each step of computation.

We point out that each neuron σ_i $(i = 0, 1, 2)$ has a synapse $(i, state)$ going to neuron σ_{state} except for neuron σ_3 (as you will see below, the difference originates from the fact that register 3 is not subject to substraction instructions); however, neuron σ_3 has a synapse $(3, b_2)$ going to neuron σ_{b_2}, which is used to stop the work of neurons σ_{b_1} and σ_{b_2} when the computation of system Π halts.

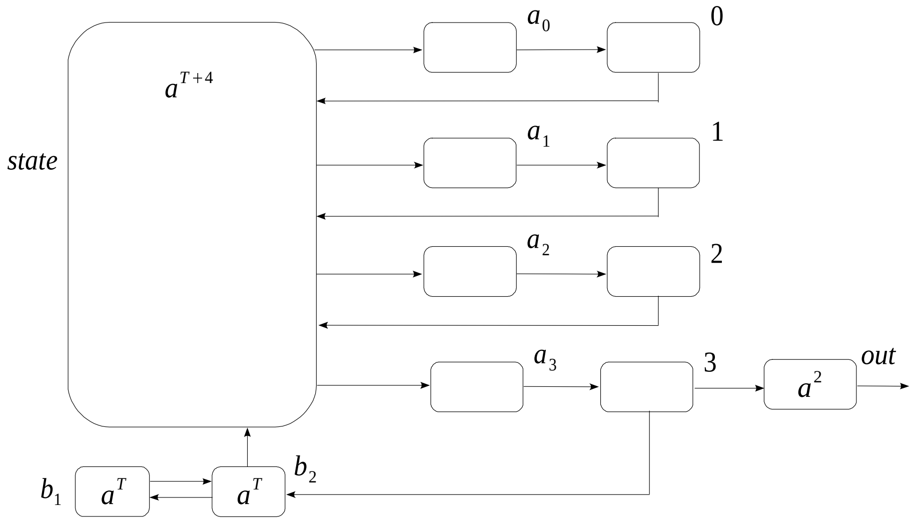


Fig. 1. The structure of system Π with the initial numbers of spikes

In system Π, each neuron is assigned with a set of rules, see Table 1, where $P(i) = 4(i+1)$, for $i = 0, 1, 2, \ldots, m$, and $T = 4(m+1)+1$. Neurons σ_i $(i = 0, 1, 2)$ have the same set of rules except of neuron σ_3, the difference originates from the fact neuron σ_3 is not subject to subtraction instruction and it is related to output the result of computation. In neuron σ_{state}, there are $m + 1$ groups of rules $R_0, R_1, \ldots, R_m$, specifically, for each ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, the set of rules $R_i = \{a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \rightarrow a^{2r+3}, a^{P(i)}(a^T)^+/a^{P(i)+T-P(k)} \rightarrow a^{2r+3}\}$ is associated; for each SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$, the set of rules $R_i = \{a^{P(i)}(a^T)^+/a^{T+3} \rightarrow a^{2r+2}, a^{P(i)-1}(a^T)^+/a^{P(i)-1+T-P(j)} \rightarrow a, a^{P(i)-2}(a^T)^+/a^{P(i)-2+T-P(k)} \rightarrow a\}$ is associated; for instruction $l_m : \mathtt{HALT}$, $R_m = \{a^{P(m)}(a^T)^+/a^{P(m)} \rightarrow a^8\}$ is associated. 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, in the initial configuration of M'_u, neuron σ_{state} has $T + 4$ spikes, which is the form $T + 4 = P(0) + T$, system Π starts to simulate the initial instruction l_0 of M'_u; with $P(m) + sT = 4(m + 1) + sT$ spikes in σ_{state}, system Π starts to output the result of computation; if the number of spikes in σ_{state} is of the form sT, then no rule in σ_{state} is enabled, which happens after the halt instruction is reached. That is why we use the label *state* for this neuron, and the function of this neuron is somewhat similar with "the finite set of states" in Turing machine.

Table 1. The rules associated with neurons in system Π

neurons	associated rules
$\sigma_{b_1}, \sigma_{b_2}$	$a^T \to a^T$
σ_i, $i = 0, 1, 2$	$a \to a$, $a(a^3)^+/a^4 \to a^2$
σ_3	$a \to a$, $a(a^3)^+/a^3 \to a^3$
$\sigma_{a_i}, i = 0, 1, 2, 3$	$a^{2i+2} \to a$, $a^{2i+3} \to a^3$, $a \to \lambda$, $a^{2j+2} \to \lambda$, $a^{2j+3} \to \lambda$, $j \in \{0, 1, 2, 3\} - \{i\}$
σ_{out}	$a \to a$, $a^3 \to \lambda$, $a^5 \to a$
σ_{state}	$R_{state} = R_0 \cup R_1 \cup \cdots \cup R_m$, where: $R_i = \{a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{2r+3},$ $a^{P(i)}(a^T)^+/a^{P(i)+T-P(k)} \to a^{2r+3}\}$, for instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$; $R_i = \{a^{P(i)}(a^T)^+/a^{T+3} \to a^{2r+2}, a^{P(i)-1}(a^T)^+/a^{P(i)-1+T-P(j)} \to a,$ $a^{P(i)-2}(a^T)^+/a^{P(i)-2+T-P(k)} \to a\}$, for instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$; $R_m = \{a^{P(m)}(a^T)^+/a^{P(m)} \to a^8\}$, for instruction $l_m : \mathtt{HALT}$

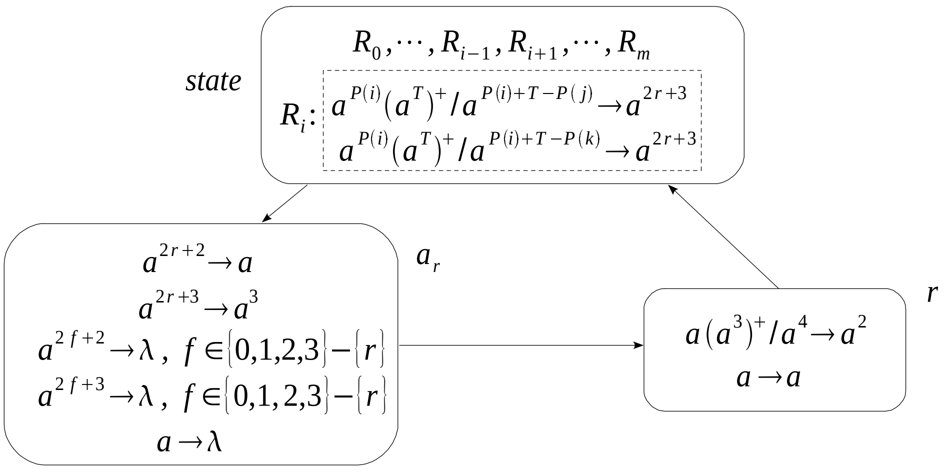
Initially, all neurons have no spike, with exception that each of neurons $\sigma_{b_1}, \sigma_{b_2}$ contains T spikes, neuron σ_{state} contains $P(0)+T = 4+T$ spikes, and neuron σ_{out} contains 2 spikes. As you will see, during the computation of M'_u, the contents of registers r, $0 \leq r \leq 3$ are encoded by the number of spikes from neuron r in the following way: if the register r holds the number $n \geq 0$, then the associated neuron σ_r will contain $3n$ spikes; the increase (resp. decrease) of the number stored in register r is simulated by adding (resp. removing) three spikes.

With T spikes inside, neurons σ_{b_1} and σ_{b_2} fire by the rule $a^T \to a^T$, sending T spikes to each other; in this way, from step 1 until system Π starting to output the result of computation (that is, until a step when neuron σ_3 fires), at each step, neuron σ_{b_2} will send T spikes to σ_{state}.

In what follows, we check the simulation of register machine M'_u by system Π, by decomposing system Π into three modules (i.e., modules ADD, SUB, and OUTPUT), and checking the work of each module.

Module ADD (Figure 2) – simulating an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$

The initial instruction of M'_u, the one with label l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, with the number of spikes being the form $P(i) + sT$ (for some $s \geq 1$) in neuron σ_{state} (in the initial configuration, neuron σ_{state} contains $P(0)+T$ spikes, and the simulation of the initial instruction with label l_0 is triggered). The



state
$R_0, \cdots, R_{i-1}, R_{i+1}, \cdots, R_m$
R_i: $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{2r+3}$
$a^{P(i)}(a^T)^+/a^{P(i)+T-P(k)} \to a^{2r+3}$
a_r
$a^{2r+2} \to a$
$a^{2r+3} \to a^3$
$a^{2f+2} \to \lambda,\ f \in \{0,1,2,3\} - \{r\}$
$a^{2f+3} \to \lambda,\ f \in \{0,1,2,3\} - \{r\}$
$a \to \lambda$
r
$a(a^3)^+/a^4 \to a^2$
$a \to a$

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

rules $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{2r+3}$ and $a^{P(i)}(a^T)^+/a^{P(i)+T-P(k)} \to a^{2r+3}$ are enabled, non-deterministically choosing one of them to be applied.

If $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{2r+3}$ is applied, then neuron σ_{state} fires, sending out $2r+3$ spikes to neurons σ_{a_i} $(i = 0, 1, 2, 3)$. Neuron σ_{a_r} sends 3 spikes to neuron σ_r by rule $a^{2r+3} \to a^3$. In neurons σ_{a_t} $(t \in \{0, 1, 2, 3\} - \{r\})$, these $2r+3$ spikes are forgotten by rule $a^{2r+3} \to \lambda$. Therefore, neuron σ_r increases its number of spikes by 3, and does not fire, which simulates the increase of the number stored in register r by 1. After consuming $P(i) + T - P(j)$ spikes by the rule $a^{P(i)}(a^T)^+/a^{P(i)+T-P(j)} \to a^{2r+3}$, the number of spikes in neuron σ_{state} is of the form $P(j) + sT$ (for some $s \geq 1$) (recalling that neuron σ_{state} receives T spikes from neuron σ_{b_2} at each step), hence system Π starts to simulate an instruction with label l_j.

Similarly, if $a^{P(i)}(a^T)^+/a^{P(i)+T-P(k)} \to a^{2r+3}$ is applied, then neuron σ_r increases its number of spikes by 3, and the number of spikes in neuron σ_{state} is of the form $P(k) + sT$ (for some $s \geq 1$). This implies that the number stored in register r is increased by 1, and system Π starts to simulate an instruction with label l_k.

The simulation of the ADD instruction is correct: we have increased the number of spikes in neuron σ_r by three, and we have passed to the simulation of one of the instructions l_j and l_k non-deterministically.

Remark: (1) The auxiliary neurons σ_{b_1} and σ_{b_2} are necessary for the function of system Π. They send T spikes to neuron σ_{state} at each step, which ensures that the number of spikes in neuron σ_{state} not less than 0.

(2) In the simulation of an ADD instruction, when neuron σ_{state} fires, it sends $2r+3$ spikes to all neurons σ_{a_i} $(i = 0, 1, 2, 3)$. Checking the rules in neurons σ_{a_i} $(i = 0, 1, 2, 3)$ (listed in Table 1), we can find that in neuron σ_{a_r} only rule $a^{2r+3} \to a$

is enabled and applied, sending three spikes to neuron σ_r; in neuron σ_{a_t} with $t \neq r$, only rule $a^{2r+3} \to \lambda$ is enabled and applied, these $2r+3$ spikes are forgotten, and neuron σ_t, $t \neq r$, receives no spike. In general, there is a bijection relation: neuron σ_r receives 3 spikes if and only if neuron σ_{state} sends out $2r+3$ spikes, where $r = 0, 1, 2, 3$. So, the neurons σ_{a_i} ($i = 0, 1, 2, 3$) work like a "sieve" such that only the register that the ADD instruction acts on can increase its number by 1.

(3) As you will see below, when a SUB instruction that acts on register r is simulated, neuron σ_{state} sends out $2r+2$ spikes. In this case, neurons σ_{a_i} ($i = 0, 1, 2, 3$) also work like a "sieve", but with different bijection relation: neuron σ_r receives 1 spike if and only if neuron σ_{state} sends out $2r+2$ spikes, where $r = 0, 1, 2, 3$.

Module SUB (Figure 3) – simulating a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$.

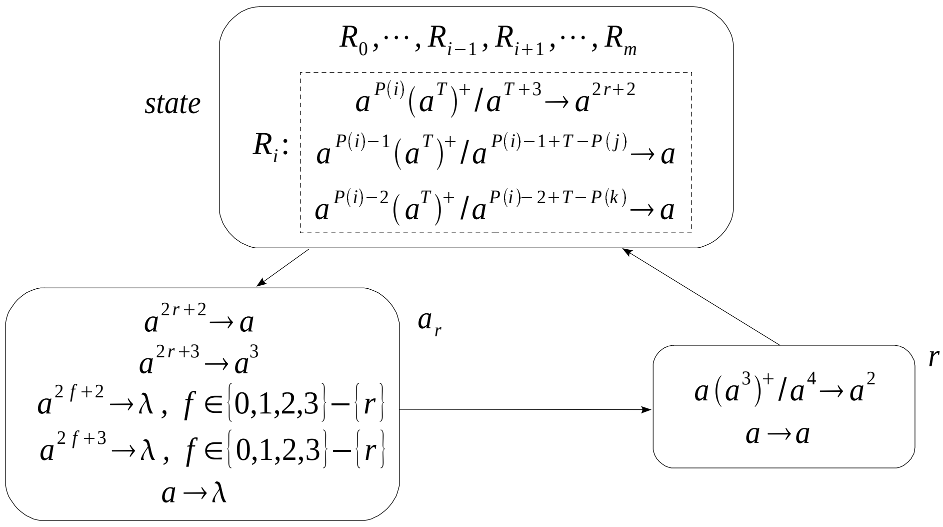

$R_0, \cdots, R_{i-1}, R_{i+1}, \cdots, R_m$
state
R_i:
$a^{P(i)}(a^T)^+/a^{T+3} \to a^{2r+2}$
$a^{P(i)-1}(a^T)^+/a^{P(i)-1+T-P(j)} \to a$
$a^{P(i)-2}(a^T)^+/a^{P(i)-2+T-P(k)} \to a$
a_r
$a^{2r+2} \to a$
$a^{2r+3} \to a^3$
$a^{2f+2} \to \lambda,\ f \in \{0,1,2,3\} - \{r\}$
$a^{2f+3} \to \lambda,\ f \in \{0,1,2,3\} - \{r\}$
$a \to \lambda$
r
$a(a^3)^+/a^4 \to a^2$
$a \to a$

Fig. 3. module SUB simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$

The execution of instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is simulated in Π in the following way. With the number of spikes in neurons σ_{state} having the form $P(i) + sT$ (for some $s \geq 1$), rule $a^{P(i)}(a^T)^+/a^{T+3} \to a^{2r+2}$ is enabled and applied, sending out $2r+2$ spikes; we suppose it is at step t. At step $t+1$, neuron σ_{a_r} spikes by the rule $a^{2r+2} \to a$, sending one spike to neuron σ_r; these $2r+2$ spikes in neuron σ_{a_t}, $t \neq r$, are forgotten by the rule $a^{2r+2} \to \lambda$ (that is, the "sieve" function of neurons σ_{a_i}, $i = 0, 1, 2, 3$, works again). For the number of spikes in neuron σ_r at step t, we consider the following two cases: (1) neuron σ_r contains at least three spikes (that is, register r is not empty); (2) neuron σ_r contains no spike (that is, register r is empty).

(1) If the number of spikes in neuron σ_r at step t is $3n$ with $n > 0$, then receiving one spike from neuron σ_{a_r} at step $t+1$; neuron σ_r has $3n+1$ spikes at step $t+2$, and rule $a(a^3)^+/a^4 \rightarrow a^2$ is enabled and applied, consuming 4 spikes, sending 2 spike to neuron σ_{state}. In this way, the number of spikes in neuron σ_r is $3(n-1)$, simulating the number stored in register r is decreased by one. After receiving these 2 spikes, the number of spikes in neuron σ_{state} is of the from $P(i) - 1 + sT$ (for some $s \geq 1$), so rule $a^{P(i)-1}(a^T)^+/a^{P(i)+T-1-P(j)} \rightarrow a$ can be applied. Consuming $P(i) - 1 + T - P(j)$ at step $t + 3$ by rule $a^{P(i)-1}(a^T)^+/a^{P(i)+T-1-P(j)} \rightarrow a$, the number of spikes in neuron σ_{state} is of the from $P(j) + sT$ (for some $s \geq 1$), which means that the next simulated instruction will be l_j. Note that this one spike emitted by neuron σ_{state} will be immediately forgotten by all neurons $\sigma_{a_0}, \ldots, \sigma_{a_3}$ at the next step because of the rule $a \rightarrow \lambda$ in these neurons.

(2) If the number of spikes in neuron σ_r at step t is 0, then at step $t + 2$, neuron σ_r contains one spike (received from neuron σ_{a_r} at step $t + 1$), and the rule $a \rightarrow a$ is applied, consuming the single spike present in neuron σ_r and sending one spike to neuron σ_{state}. Neuron σ_{state} contains $P(i) - 2 + sT$ (for some $s \geq 1$) spikes at step $t + 3$, rule $a^{P(i)-2}(a^T)^+/a^{P(i)+T-2-P(k)} \rightarrow a$ is enabled and applied, consuming $P(i)-2+T-P(k)$ spikes. So, the number of spikes in neuron σ_{state} is of the from $P(k) + sT$ (for some $s \geq 1$), and 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.

Module OUTPUT (Figure 4) – outputting the result of computation.

Assume now that the computation in M_u' halts, which means that the halt instruction l_m is reached. For system Π, this means that neuron σ_{state} contains $P(m)+sT$ spikes (for some $s \geq 1$). At that moment, neuron σ_3 contains $3n$ spikes, for n being the content of register 3 of M_u'. Having $P(m)+sT$ spikes inside, neuron σ_{state} gets fired and emits 8 spikes by the rule $a^{P(m)}(a^T)^+/a^{P(m)} \rightarrow a^8$. After that, the number of spikes in neuron σ_{state} is of the form sT (for some $s \geq 1$), no rule can be applied anymore in neuron σ_{state}.

At the next step, neurons σ_{a_0}, σ_{a_1}, σ_{a_2} forget these 8 spikes received from σ_{state} by the rule $a^8 \rightarrow \lambda$; only neuron σ_{a_3} sends one spike to neuron σ_3 by the

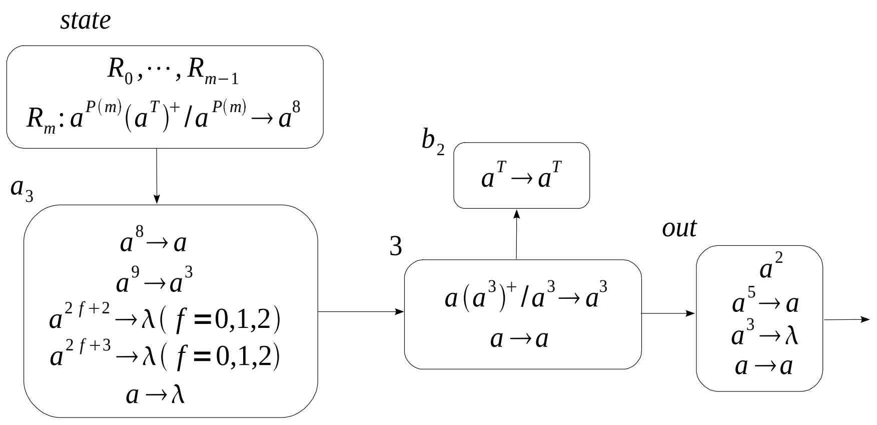


Fig. 4. Module OUTPUT

rule $a^8 \to a$. In this way, neuron σ_3 has $3n+1$ spikes, hence the rule $a(a^3)^+/a^3 \to a^3$ can be applied, sending three spikes to neuron σ_{out}. With five spikes inside (three spikes were received from neuron σ_3; two spikes were contained from the initial configuration), neuron σ_{out} fires by the rule $a^5 \to a$, which is the first spike sent out by system Π to the environment. Let t be the moment when neuron σ_{out} fires.

When neuron σ_3 spikes at step $t-1$, neuron σ_{b_2} also receives 3 spikes from neuron σ_3, which gets "over flooded" and is blocked. So, neurons σ_{b_1} and σ_{b_2} stop their works.

Note that at step t, neuron σ_3 contains $3(n-1)+1$ spikes (three spikes were already consumed at step $t-1$). From step t on, at each step, three spikes are consumed in neuron σ_3 by the rule $a(a^3)^+/a^3 \to a^3$, sending 3 spikes to neuron σ_{out}; these three spikes in neuron σ_{out} are forgotten by the rule $a^3 \to \lambda$. So, at step $t+(n-1)$, neuron σ_3 contains one spike, and the rule $a \to a$ is enabled and applied, sending one spike to neuron σ_{out}. With one spike inside, neuron σ_{out} fires for the second (and last) time by the rule $a \to a$ at step $t+n$. The interval between these two spikes sent out to the environment by the system is $(t+n)-t=n$, which is exactly the number stored in register 3 of M'_u at the moment when the computation of M'_u halts.

From the above description, it is clear that the register machine M'_u is correctly simulated by system Π. Therefore, Theorem 1 holds.

4 Conclusions and Remarks

In this note, a new way is introduced for simulating register machines by SN P systems, where a neuron works like "the finite set of states" in Turing machine. By this new way, we can use less neurons to construct universal SN P systems. Specifically, a universal system with extended rules (without delay) having 12 neurons is constructed.

In the universal SN P system Π constructed in Section 3, four neurons are associated with 4 registers; one neuron is used to output the result of computation; one neuron is used for all instructions of a register machine; 2 auxiliary neurons are used to feed spikes at each step; 4 auxiliary neurons are used between the neuron associated with all instructions and neurons associated with registers, which work as a "sieve". 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".

In this note, we only considered SN P systems with extended rules without delay. Can we extend this way to the case of SN P systems with standard rules (a little more neurons seems necessary), asynchronous SN P systems, or other variants and modes of SN P systems?

The universal SN P system constructed in this note is already quite small. If we start from universal register machines to construct universal SN P systems, then it may be not easy to get significant improvement. Of course, it is still possible to have smaller universal SN P systems, if we start construction from other small universal computational devices.

Acknowledgements. The comments from three anonymous referees are greatly acknowledged. The work was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60533010), 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 and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae*, 2006, 71(2–3): 279–308
2. I. Korec, Small universal register machines, *Theoretical Computer Science*, 1996, 168: 267–301
3. M. Minsky, *Computation – Finite and Infinite Machines*, Prentice Hall, New Jersey, 1967
4. A. Păun, Gh. Păun. Small universal spiking neural P systems, *BioSystems*, 2007, 90(1): 48–60
5. Gh. Păun, *Membrane Computing – An Introduction*, Springer-Verlag, Berlin, 2002
6. Gh. Păun, G. Rozenberg, A. Salomaa, eds., *Handbook of Membrane Computing*, Oxford University Press, 2010
7. Y. Rogozhin, Small universal Turing machines, *Theoretical Computer Science*, 1996, 168: 215–240
8. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997
9. X. Zhang, X. Zeng, L. Pan, Smaller universal spiking neural P systems, *Fundamental Informaticae*, 2008, 87(1): 117–136
10. The P System Web Page: http://ppage.psystems.eu