Fundamenta Informaticae 97 (2009) 275–294
DOI 10.3233/FI-2009-200
IOS Press

Homogeneous Spiking Neural P Systems

Xiangxiang Zeng, Xingyi Zhang, Linqiang Pan[*]
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
xzeng@foxmail.com, xyzhanghust@gmail.com, lqpan@mail.hust.edu.cn

Abstract. Spiking neural P systems are a class of distributed parallel computing models inspired from the way the neurons communicate with each other by means of electrical impulses (called "spikes"). In this paper, we consider a restricted variant of spiking neural P systems, called homogeneous spiking neural P systems, where each neuron has the same set of rules. The universality of homogeneous spiking neural P systems is investigated. One of universality results is that it is sufficient for homogeneous spiking neural P system to have only one neuron that behaves non-deterministically in order to achieve Turing completeness.

Keywords: Membrane computing, Spiking neural P system, Register machine.

1. Introduction

Biological neural networks are the result of millions of years of evolution, which lead to a very intricate information-processing structure and functioning, and this is a rich source of inspiration for computer scientists to design various computational devices and intelligent machines. As it is known, the two very important computing devices, Turing machine and finite automaton, were inspired from the brain activity. Currently, neural computing based on spiking is a field that is being heavily investigated (see, e.g., [4, 9, 10]). Recently, a new class of such computing devices was developed, that is *spiking neural P*

[*]Address for correspondence: Huazhong University of Science and Technology, Wuhan 430074, Hubei, People's Republic of China

systems (in short, SN P systems) [7]. SN P systems are a variant of tissue-like and neural-like P systems with specific ingredients and way of functioning, bringing spiking neurons into membrane computing.

Informally, in SN P systems, the neurons are 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. Every cell contains 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 cells. The applicability of each rule is determined by checking the content of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use some of its rules then one of such rules must be used. The rule to be applied is non-deterministically chosen. Thus, the rules are used in a sequential manner in each neuron, but neurons function in parallel with each other. Observe that, as it usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. When a cell sends out spikes it becomes “closed” (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot “fire” (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike reaches its target cell. In SN P systems, this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a forgetting rule, that removes from the neuron a predefined number of spikes.

Many computational properties of SN P systems have already been investigated; for example, SN P systems have been proved to be Turing-complete when considered as number computing devices [7], when used as language generators [2, 3], and also when computing functions [14].

A neuron (in the initial configuration of an SN P system) is characterized by the number of spikes present in it, its associated set of rules, and its outdegree and indegree in the synapse graph of the system. It is of a clear interest to restrict some of the above features of SN P systems without loss of computational completeness. Actually, some normal forms are already given, e.g. [5, 6, 12]. In this paper, we focus on the restriction that each neuron has the same set of rules. SN P systems with such restriction are called *homogeneous spiking neural P systems* (in short, HSN P systems). Two kinds of HSN P systems are considered: one has weighted synapses, another one has usual synapses (i.e., without weight on synapses). We prove that both of these two kinds of HSN P systems are universal.

Homogeneity appears in several computational devices in computer science, for example, classical cellular automata are composed of simple and homogenous units. Consideration of homogeneousness in SN P systems has also a clear biological motivation. In the (human) brain, there are about 100 billion neuron cells. Most of the neurons are very similar to each other, they are “designed” by nature to have the same “set of rules ”, “working” in a uniform way to transform input into output. However, in the usual SN P systems, neurons are “different” from each other in the sense of having different rules. So, from this point of view, homogeneous SN P systems are closer to biological reality than the usual SN P systems, and in this sense, the universality results given in this paper are interesting.

The rest of this paper is organized as follows. In the next section, we introduce some necessary prerequisites. Section 3 gives the definition of spiking neural P systems. In Section 4, we prove the universality of HSN P systems as devices generating sets of numbers, and in Section 5, the case of universal HSN P systems in accepting mode is investigated. Conclusions and remarks are drawn in Section 6.

2. Prerequisites

The reader is assumed to have some familiarity with (basic elements of) language theory, e.g., from [17], as well as basic membrane computing [15] (for more updated information about membrane computing, please refer to [19]), so we directly introduce some notations and basic definitions here.

For an alphabet V, 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 simply a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$. The family of recursively enumerable languages is denoted by RE. The family of Turing computable sets of natural numbers is denoted by NRE.

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 $M = (m, H, l_0, l_h, I)$, 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),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (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 computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by labels (and made possible by the content of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE (see, e.g. [11]).

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its content.

We can also use a register machine in the accepting mode: a number is stored in the first register (all other registers are empty); if the computation starting in this configuration eventually halts, then the number is accepted. Again, all sets of numbers in NRE can be obtained, even using deterministic register machines, i.e., with the ADD instructions of the form $l_i : (\mathtt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\mathtt{ADD}(r), l_j)$).

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

3. Spiking Neural P Systems

We start by introducing the basic class of SN P systems – SN P systems without weight on synapses and without restriction on the set of rules. For more details on these systems, please refer to [7].

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 $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, 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) $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$;

      (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ 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.

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 rule $E/a^c \rightarrow a; d \in R_i$ can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i), the neuron is fired, and it produces a spike 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 the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used at step t and $d \geq 1$, then at 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 a spike along it, then the particular spike is lost). At step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can apply rules again).

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

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

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

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, $E_1/a^{c_1} \rightarrow a; d_1$ and $E_2/a^{c_2} \rightarrow a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, 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 at 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 determined by checking the total number of spikes contained in the neuron against a regular set associated with the rule.

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 from 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. With any computation, halting or not, we associate a *spike train*, the binary sequence with occurrences of 1 indicating time instances when the output neuron sends a spike out of the system (we also say that the system itself spikes at that time).

The *result of a computation* can be defined in several ways. In this paper, with any spike train containing at least two spikes, the first two being emitted at steps t_1, t_2, one associates a result, in the form of number $t_2 - t_1$; we say that this number is computed by Π. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript indicates that we only consider the distance between the first two spikes of any computation; note that 0 cannot be computed, that is why we disregard this number when estimating the computing power of any device).

SN P systems can also work in the accepting mode: we start the computation from an initial configuration, and we introduce in the input neuron two spikes, at steps t_1 and t_2; the number $t_2 - t_1$ is accepted by the system if the computation eventually halts. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π.

An SN P system such that each neuron has the same set of rules is said to be *homogenous* (in short, we say HSN P system). That is, $R_1 = R_2 = \cdots = R_m$ for neurons $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, in the system Π.

A *semi-homogenous SN P system* (in short, SHSN P system) is an SN P system such that all neurons have the same set of rules except for one neuron.

SN P systems with *weighted synapses* [13] have the same components and the similar way of working as the above basic SN P systems except for the synapses, for which we have

$$syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times \mathbb{N}.$$

For a synapse $(i, j, r) \in syn$, if a spike is emitted by neuron σ_i, then r spikes will be received by neuron σ_j. Clearly, basic SN P systems are a special case of SN P systems with weighted synapses, where the weight of each synapse is one.

We denote by $N_\alpha HSNP(weight_k)$ the families of all sets $N_\alpha(\Pi)$ computed by HSN P systems with the weight on synapses at most k, where $\alpha \in \{2, acc\}$; if the weight on synapses is always one (as the synapses in basic SN P systems), then we remove the indication $weight$ from the notation.

4. HSN P Systems Working in the Generating Mode

In this section, we investigate the universality of HSN P systems as number generating devices. First, we consider the case of HSN P systems with weighted synapses, and prove they are universal. As we will see, the weight on synapses significantly simplifies the construction and structure of an HSN P system simulating a register machine. Then, as expected, the weight on synapses is removed by replacing the corresponding neurons with several copies of them, and in this way, we prove that HSN P systems with usual synapses are also universal. Note that the universality of HSN P systems with usual synapses is obtained by modifying an HSN P systems with weighted synapses simulating a register machine. At this moment, we do not know how to directly construct an HSN P system with usual synapses to simulate a register machine.

As it is usual, HSN P systems are represented graphically, which may be easier to understand than in a symbolic way. We use an oval with rules inside to represent a neuron, and a directed graph to represent the structure of the HSN P system: the neurons are placed in the nodes of the graph and the 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. Because each neuron in an HSN P system has the same set of rules, the rules are omitted in the representation of the HSN P system. When the weight on a synapse is one, it is also omitted in the graph.

4.1. HSN P Systems with Weighted Synapses

Theorem 4.1. $NRE = N_2HSNP(weight_5)$.

Proof:
We show that $NRE \subseteq N_2HSNP(weight_5)$; the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). Let us consider a register machine $M = (m, H, l_0, l_h, I)$. Without any loss of generality, we may assume that in the halting configuration, all registers of M different from register 1 are empty, and that the register 1 is never decremented during the computation, we only add to its content. In what follows, a specific HSN P system Π will be constructed to simulate the register machine M.

In the HSN P system Π, each neuron has the same set of rules $R = \{a \to \lambda, a^2 \to a, a^3 \to a, a^3 \to a; 1, a^4 \to \lambda, a^2(a^5)^+/a^7 \to a; 1, a^4(a^5)^+/a^5 \to a\}$, which is shown in Figure 1. The system Π consists of three types of modules – ADD module, SUB module, and FIN module shown in Figures 2, 3, 4, respectively. Of course, each module is composed of neurons as in Figure 1.

In general, for each register r of M, a neuron σ_r is associated whose content corresponds to the content of the register. Specifically, if register r holds the number $n \geq 0$, then neuron σ_r will contain $5n$ spikes. For each label l_i of an instruction in M, a neuron σ_{l_i} is associated. Initially, all neurons are empty, with the single exception of neuron σ_{l_0} associated with the initial instruction l_0 of M, which contains exactly two spikes. During the computation, a neuron σ_{l_i} which receives 2 spikes will become active and start to simulate an instruction $l_i : (\mathtt{OP}(r), l_j, l_k)$ of M: starting with neuron σ_{l_i} activated, operating the register as requested by OP, then introducing 2 spikes in one of the neurons σ_{l_j}, σ_{l_k}, which becomes in this way active. When activating neuron σ_{l_h}, associated with the halting label of M, the computation in M is completely simulated in Π; we will have the output neuron spiking twice, at an interval of time which corresponds to the number stored in register 1 of M.

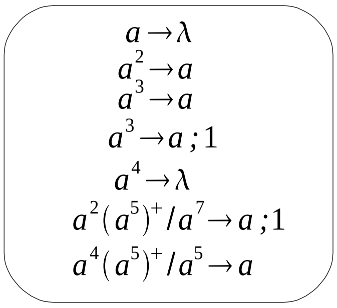

Figure 1. Neurons in HSN P system Π

Before describing these modules and their work, let us recall that the labels are injectively associated with the instructions of M, hence each label precisely identifies one instruction, either an ADD or a SUB one, with the halting label having a special situation – it will be dealt with by the FIN module; the register 1 is never decremented.

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

Module ADD is composed of seven neurons: neuron σ_r for register r, neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ for instructions with labels l_i, l_j, l_k, and three auxiliary neurons $\sigma_{a_{i1}}, \sigma_{a_{i2}}, \sigma_{a_{i3}}$, as shown in Figure 2.

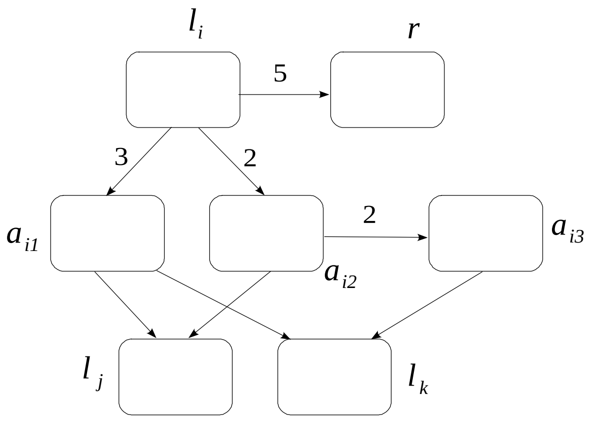

Figure 2. Module ADD for simulating instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$

The initial instruction in register machine M, labeled with l_0, is an ADD instruction. Let us assume that we are at a step when we have to simulate an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, with two spikes present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neuron, except in those neurons associated with registers. Having two spikes inside, neuron σ_{l_i} gets fired by rule $a^2 \to a$. Simultaneously, neurons $\sigma_{a_{i1}}$, $\sigma_{a_{i2}}$ and σ_r receive 3 spikes, 2 spikes and 5 spikes, respectively (the spike sent out by neuron σ_{l_i} is amplified by the weighted synapses). In this way, the content of neuron σ_r increases by 5, thus simulating the increase of the number stored in register r by 1.

At the next step, the computation of M passes non-deterministically to one of the instructions with labels l_j and l_k; that is, we have to ensure the firing of neurons σ_{l_j} or σ_{l_k} in system Π, non-deterministically choosing one of them. To this aim, we use the non-determinism of the rules $a^3 \to a$ and $a^3 \to a; 1$. Because neuron $\sigma_{a_{i1}}$ received 3 spikes at the last step, it has to choose non-deterministically one rule between $a^3 \to a$ and $a^3 \to a; 1$ to apply. There are the following two cases.

(1) If rule $a^3 \to a$ is applied, then both neurons σ_{l_j} and σ_{l_k} receive a spike from neuron $\sigma_{a_{i1}}$. For neuron σ_{l_k} this is the unique spike it receives now, which is consumed immediately by the forgetting rule $a \to \lambda$. In the next step, neuron σ_{l_k} receives another spike from neuron $\sigma_{a_{i3}}$, which is also consumed by the forgetting rule $a \to \lambda$ after one more step. However, neuron σ_{l_j} receives two spikes at one step, one from neuron $\sigma_{a_{i1}}$ and another one from neuron $\sigma_{a_{i2}}$, hence in the next step it is fired.

(2) If rule $a^3 \to a; 1$ is applied (note that the only difference with the rule $a^3 \to a$ is the time of delay), then neuron σ_{l_j} receives one spike from neuron $\sigma_{a_{i2}}$ which is forgotten by the rule $a \to \lambda$ in the next step; the spike received in the next step from neuron $\sigma_{a_{i1}}$ is also forgotten after one more step. However, in the next step neuron σ_{l_k} receives two spikes: one from neuron $\sigma_{a_{i1}}$, another one from neuron $\sigma_{a_{i3}}$, so neuron σ_{l_k} is activated (the rule $a^2 \to a$ is enabled).

Therefore, from firing neuron σ_{l_i}, we pass to firing non-deterministically one of neurons σ_{l_j}, σ_{l_k}, which correctly simulates the ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$.

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

Module SUB, shown in Figure 3, is composed of six neurons: neuron σ_r for register r, neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ for instructions with labels l_i, l_j, l_k, and two auxiliary neurons $\sigma_{b_{i1}}, \sigma_{b_{i2}}$.

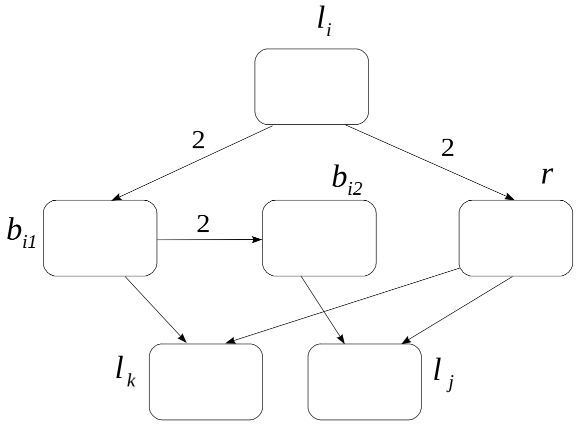

Figure 3. Module SUB for simulating instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$

Instruction l_i is simulated in Π in the following way. Initially, there are two spikes in neuron σ_{l_i}, and no spike in other neurons, except neuron σ_r. Let t be the moment when neuron σ_{l_i} fires. At step t, neurons $\sigma_{b_{i1}}$ and σ_r receive two spikes. At step $t+1$, neurons $\sigma_{b_{i1}}$ fires, neurons $\sigma_{b_{i2}}$ receives two spikes and σ_{l_k} receives one spike. At step $t+2$, neuron σ_{l_j} receives one spike from neuron $\sigma_{b_{i2}}$. For neuron σ_r, there are the following two cases.

(1) The number of spikes in neuron σ_r is 0 at step t. Then at step $t+1$, the two spikes received from neuron σ_{l_i} are consumed by rule $a^2 \to a$, sending one spike to neurons σ_{l_j} and σ_{l_k}. At step $t+2$,

neuron σ_{l_k} contains 2 spikes (one from neuron σ_r, another one from $\sigma_{b_{i1}}$) and fires (starting the next simulation of instruction with label l_k); neuron σ_{l_j} forgets the spike received from neuron σ_r, at the same time, it receives one spike from neuron $\sigma_{b_{i2}}$ that is forgotten after one more step.

(2) The number of spikes in neuron σ_r is $5n$ ($n > 0$) at step t. Then, rule $a^2(a^5)^+/a^7 \rightarrow a; 1$ consumes 7 spikes present in neuron σ_r, and at step $t+2$ neurons σ_{l_j} and σ_{l_k} receive one spike from neuron σ_r. In this case, neuron σ_{l_k} forgets the two spikes received from neurons $\sigma_{b_{i1}}$ and σ_r in two consecutive steps (one spike in each step, and forgotten by the rule $a \rightarrow \lambda$); neuron σ_{l_j} gets 2 spikes at step $t+2$ and fires, starting the simulation of instruction with label l_j.

Note that there is an interference between SUB modules. When simulating an instruction l_i : $(\texttt{SUB}(r), l_j, l_k)$, neuron σ_r sends one spike to all neurons σ_{l_u}, σ_{l_v} from modules associated with instructions l_s : $(\texttt{SUB}(r), l_u, l_v)$ (that is, subtracting from the same register r); however, they are forgotten by the rule $a \rightarrow \lambda$ immediately. So, no undesired effect appears.

Module FIN – outputting the result of computation.

Module FIN is composed of four neurons: one for the halting instruction l_h, one for the register 1, one auxiliary neuron σ_{c_1}, and one output neuron σ_{out}, as shown in Figure 4.

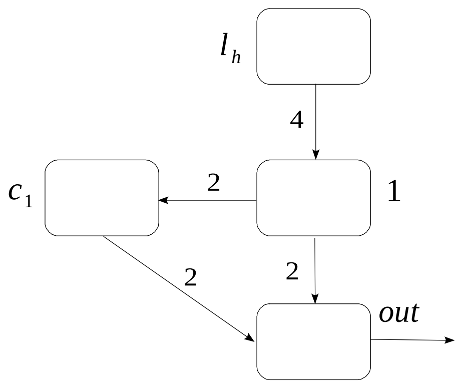

Figure 4. Module FIN (outputting the result of the computation)

Assume now that the computation in M halts, which means that the halting instruction is reached. This means that neuron σ_{l_h} in Π gets two spikes and fires by rule $a^2 \rightarrow a$. At that moment, neuron σ_1 contains $5n$ spikes, for the number n stored in register 1 of M. When σ_{l_h} fires, neuron σ_1 receives 4 spikes because of synapse weight; in this way, the number of spikes in neuron σ_1 becomes $5n+4$ spikes. Rule $a^4(a^5)^+/a^5 \rightarrow a$ is enabled and applied, neurons σ_{out} and σ_{c_1} receive two spikes from neuron σ_1; we suppose it is at step t. At step $t+1$, neuron σ_{out} fires, the system Π sends the first spike to the environment. From step $t+1$ to step $t+n-1$, at each step, neuron σ_{out} receives 4 spikes: two spikes from neuron σ_{c_1}, the other two spikes from neuron σ_1, then these four spikes are consumed by rule $a^4 \rightarrow \lambda$ in the next step. So from step $t+2$ to step $t+n$, the system sends no spike to the environment. In neuron σ_1, at step t, 5 spikes are consumed; from step $t+1$ to step $t+n-1$, at each step, 5 spikes are consumed; so at step $t+n$, neuron σ_1 contains 4 spikes that are consumed by rule $a^4 \rightarrow \lambda$, sending no spike out. Hence, at step $t+n$, neuron σ_{out} receives only 2 spikes, which are from neuron σ_{c_1}. At step $t+n+1$, neuron σ_{out} spikes again, the system sends the second spike to the environment. The interval between these two spikes sent out to the environment by the system is $(t+n+1)-(t+1)=n$, which is exactly the number stored in register 1 of M at the moment when the computation of M halts.

From the above description of the modules and their work, it is clear that the register machine M is correctly simulated by the HSN P system Π. Therefore, $N_2(\Pi) = N(M)$ and this completes the proof. □

From the proof of Theorem 4.1, we can find that in the system Π, only auxiliary neurons $\sigma_{a_{i1}}$ associated with ADD instructions behave non-deterministically. In what concerns the number of neurons behaving non-deterministically, we can have the following result.

Corollary 4.1. Each set from NRE can be computed by an HSN P system with weighted synapses such that only one neuron behaves non-deterministically, although all neurons have rules that can be used non-deterministically.

Proof:
It is easy to observe that the only neurons behaving non-deterministically are $\sigma_{a_{i1}}$ in modules ADD. Let us "unify" all these neurons for all ADD instructions: (1) remove all neurons $\sigma_{a_{i1}}$ from system Π and their associated synapses, (2) add a neuron σ_{a_1} with synapses (l_i, a_1) and associated weight 3 for all instructions $l_i : (\texttt{ADD}(r), l_j, l_k)$ in the register machine we want to simulate, and synapses (a_1, l) and associated weight 1 for all $l \in H$. When starting to simulate an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, neuron σ_{a_1} receives 3 spikes, and either spikes immediately, or in the next step. The spike is sent to all neurons σ_l without amplification, where $l \in H$. It meets another spike only in one neuron: σ_{l_j} if σ_{a_1} spikes immediately, or σ_{l_k} if σ_{a_1} spikes in the next step. In all neurons different from these neurons, the spike is forgotten. The system obtained in this way is clearly equivalent with the one constructed in the proof of Theorem 4.1 and it contains only one neuron that behaves non-deterministically. □

In the system given in the proof of Corollary 4.1, the only rules that can be used non-deterministically are $a^3 \to a$ and $a^3 \to a; 1$, and the non-determinism only appears in neuron σ_{a_1}. If we replace neuron σ_{a_1} by the neuron from Figure 5(a), and replace all other neurons in the system by neuron from Figure 5(b), keeping the synapses and their associated weight unchanged, then the resulted system is clearly equivalent with the one constructed in the proof of Corollary 4.1. So we can reformulate Corollary 4.1 as follows.

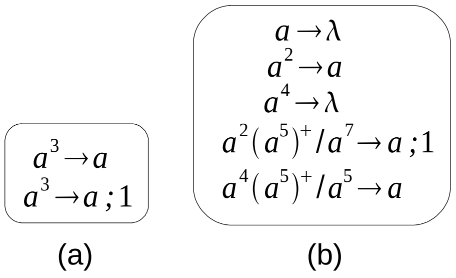

Figure 5. Neurons in an SHSN P system Π_1

Corollary 4.2. Each set from NRE can be computed by an SHSN P system with weighted synapses having only one neuron with rules that can be used non-deterministically.

In all previous constructions simulating register machines, there are two rules with delay, $a^3 \to a; 1$ and $a^2(a^5)^+/a^7 \to a; 1$. Actually, the delay in rule $a^2(a^5)^+/a^7 \to a; 1$ can be removed with the price that the number of rules and the number of neurons slightly increase.

Corollary 4.3. Each set from NRE can be computed by an SHSN P system with weighted synapses having only one neuron that has a rule with delay and whose rules can be used non-deterministically.

Proof:
For a given register machine M, by modifying the previous constructions, we can obtain an SHSN P system Π_2 with properties described in Corollary 4.3 to simulate M. In system Π_2, neuron σ_{a_1} is as shown in Figure 6(a), all other neurons are as shown in Figure 6(b). System Π_2 consists of three types of modules – module ADD, module SUB, module FIN, which are shown in Figures 7, 8 and 9, respectively. All modules ADD share neuron σ_{a_1}. Initially, all neurons are empty except that neuron σ_{l_0} contains two spikes. The work of system Π_2 can be checked in a similar way to previous systems, so we shall not explain it. □

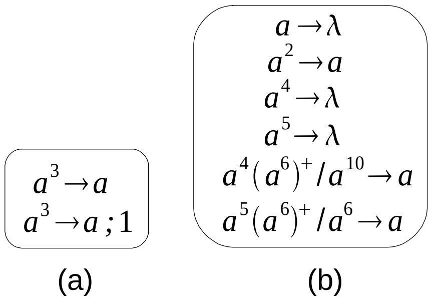

Figure 6. Neurons in an SHSN P system Π_2

4.2. HSN P Systems with Usual Synapses

We now pass to considering the universality of HSN P systems with usual synapses (i.e., there is no weight on synapses). As expected, the weight on synapses of the universal system constructed in the previous subsection can be removed at the price of some more neurons.

For an SN P system Π with standard rules and weighted synapses without the restriction that each neuron has the same set of rules, the weight on synapses can easily be removed in the following way (see Figure 10): (1) all neurons keep unchanged; (2) each synapse (i, j) with weight k ($k \geq 1$) is replaced by k neurons with usual synapses from neuron σ_i to all these k neurons and usual synapses from all these k neurons to neuron σ_j, where these k neurons have only one spiking rule $a \to a$. The resulted system is denoted by Π'. If system Π can correctly simulate a register machine, then it is not difficult to see that the system Π' can also correctly simulate this register machine. However, this strategy of removing weight on synapses does not seem to work for the case of HSN P systems, because each neuron in the system constructed in the proof of Theorem 4.1 has rule $a \to \lambda$.

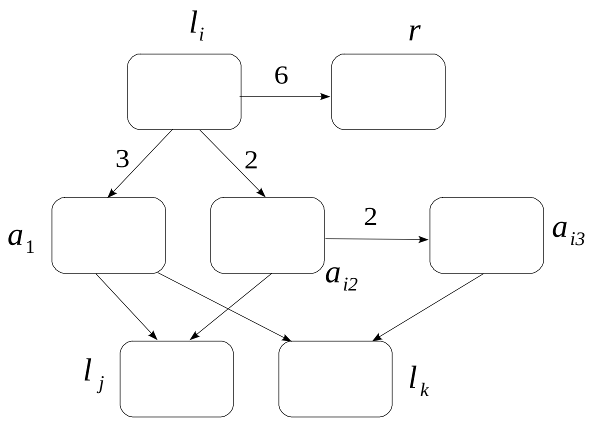

Figure 7. Module ADD in an SHSN P system Π_2

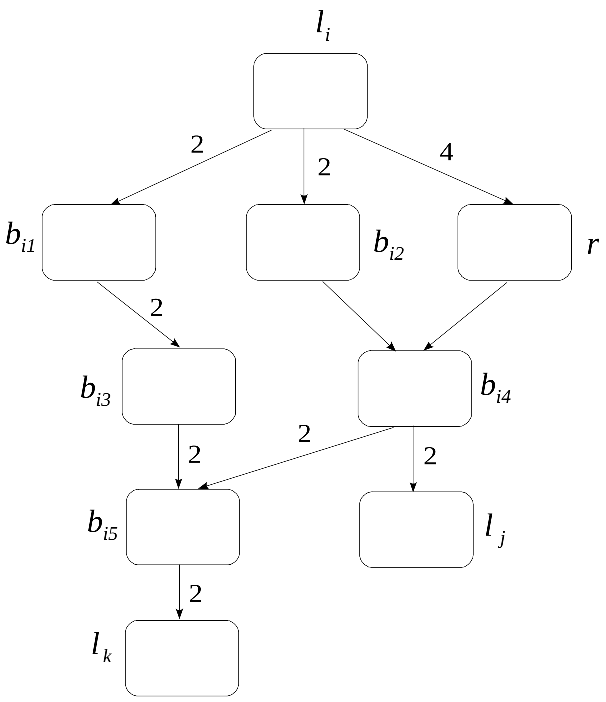

Figure 8. Module SUB in an SHSN P system Π_2

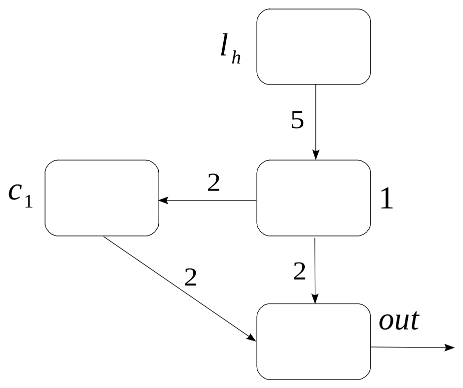

Figure 9. Module FIN in an SHSNP system Π_2

In what follows, we give a direct construction of an HSN P system which simulates a register machine, by modifying the construction used in the previous section.

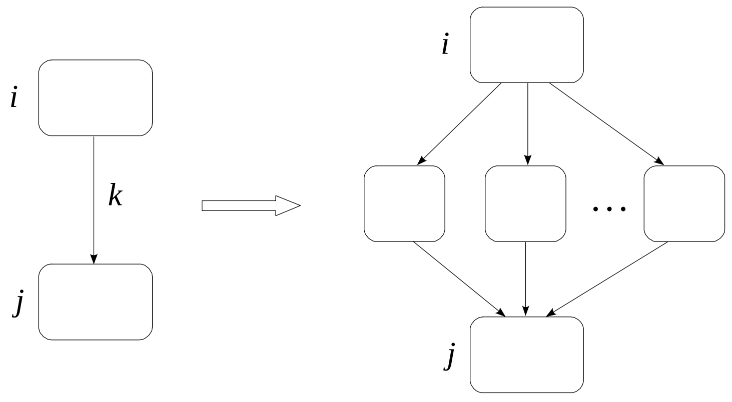

Figure 10. Removing weight on a synapse

Theorem 4.2. $NRE = N_2HSNP$.

Proof:
For a register machine $M = (m, H, l_0, l_h, I)$, we construct an HSN P system Π' to simulate it, by modifying the system Π constructed in the proof of Theorem 4.1. Specifically, the system Π' is also composed of neurons shown in Figure 1 (the same neuron as that in Π in the previous section). The three types of modules of Π' are respectively obtained by modifying the three types of modules of Π.

In general, if l_i is the label of an ADD instruction, then the associated neuron σ_{l_i} in Π is replaced by 5 neurons, if l_i is the label of a SUB instruction, then the associated neuron σ_{l_i} in Π is replaced by 2 neurons, if l_i is the label of halting instruction, then the associated neuron σ_{l_i} in Π is replaced by 4 neurons. For system Π', a dash box with label l_i is used to denote the group of neurons that are associated with instruction l_i of M, where the number of neurons in the dash box depends on the type of instruction

l_i as described above. Neurons associated with registers keep unchanged. In each module, some auxiliary neurons are also replaced by a group of neurons, and synapses between neurons are specified.

For an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, ADD module in Figure 2 is modified in the following way: (1) removing the weight on synapses; (2) replacing the neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ by corresponding groups of neurons, where the group associated with l_i has 5 neurons, the number of neurons in the groups associated with l_j and l_k depends on the types of instructions l_j and l_k; (3) replacing the auxiliary neuron $\sigma_{a_{i2}}$ by two neurons; (4) neurons $\sigma_r, \sigma_{a_{i1}}, \sigma_{a_{i3}}$ keep unchanged; (5) specifying synapses between neurons, there is a synapse from each neuron in group l_i to neuron σ_r, neuron $\sigma_{a_{i1}}$ has three synapses coming from three neurons of group l_i and synapses going to each neuron in groups l_j and l_k, each neuron in group a_{i2} has two synapses coming from two neurons of group l_i and one synapse going to neuron $\sigma_{a_{i3}}$, neuron $\sigma_{a_{i3}}$ has synapses going to each neuron in group l_k. The resulted ADD module for system Π' is shown in Figure 11.

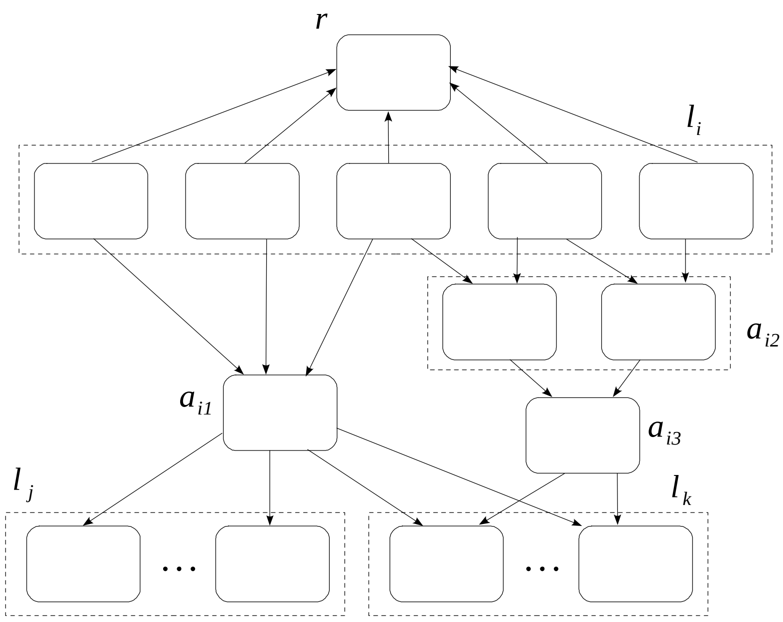

Figure 11. Module ADD for system Π'

The SUB module and FIN module from Figures 3 and 4 can be modified in a similar way to ADD module. The resulted SUB module and FIN module for system Π' are shown in Figures 12 and 13, respectively. We shall not go into details of the modification of SUB module and FIN module, because it is not difficult to understand the corresponding structure from the representations in Figures 12 and 13.

In the initial configuration of the system Π', each of the five neurons in the group l_0 has two spikes, and all other neurons are empty. We shall not explain the details about the work of each module, since they can be checked in a similar way as in the proof of Theorem 4.1. In what follows, we just point out that all neurons in a group have the same behavior.

Note that in system Π, the non-determinism of ADD instruction is simulated by the non-deterministic choice of rules $a^3 \to a$ and $a^3 \to a; 1$, this happens in the auxiliary neuron σ_{a_1}, and neuron σ_{a_1} is the unique neuron that possibly contains 3 spikes. Neuron σ_{a_1} in system Π' has the same function as in

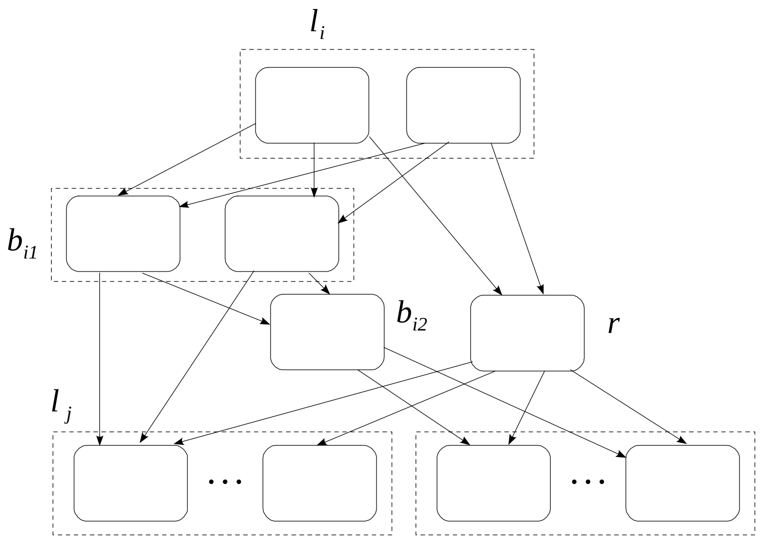

Figure 12. Module SUB for system Π'

system Π. This fact means that all the neurons except neuron σ_{a_1} in system Π' work in a deterministic way. In particular, neurons in group l_0 have the same behavior – all of them spike by the rule $a^2 \to a$, when the simulation starts; from then on, all neurons in a group have the same behavior. So register machine M is correctly simulated by the HSN P system with usual synapses Π'. $\square$

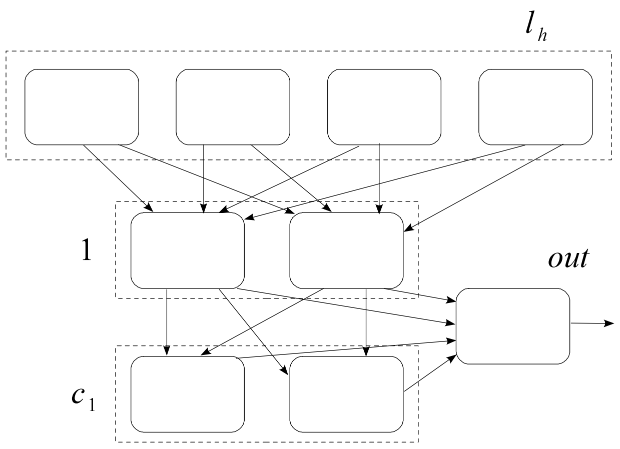

Figure 13. Module FIN for system Π'

Following the approach in the proof of Theorem 4.2, the weight on synapses in the systems of Corollaries 4.1 – 4.3 can also be removed. The corresponding results are given as follows.

Corollary 4.4. Each set from NRE can be computed by an HSN P system with usual synapses such that only one neuron behaves non-deterministically, although all neurons have rules that can be used non-deterministically.

Corollary 4.5. Each set from NRE can be computed by an SHSN P system with usual synapses having only one neuron that has a rule with delay and whose rules can be used non-deterministically.

5. HSN P Systems Working in the Accepting Mode

An HSN P system can also be used in the accepting mode, which means working in the following way: the special neuron σ_{in} is used as an input neuron, which can receive spikes from the environment; we assume that exactly two spikes are entering the system; the number n of steps elapsed between the two spikes is the one analyzed, in the sense that, if, after receiving the two spikes, the system halts (not necessarily in the moment of receiving the second spike), then number n is accepted.

Theorem 5.1. $NRE = N_{acc}HSNP(weight_6)$.

Proof:
The proof is a direct consequence of the proof of Theorem 4.1. We start from a deterministic register machine $M = (m, H, l_0, l_h, I)$ and construct the SN P system Π as in the proof of Theorem 4.1 – with changes which will be immediately mentioned, as well as a further module, called INPUT, which takes care of initializing the work of Π. The system Π is composed of neurons from Figure 1.

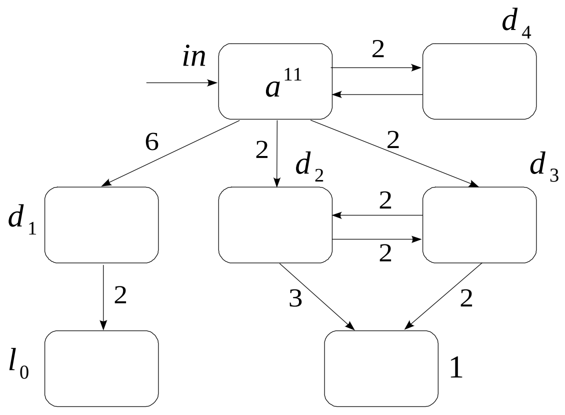

Figure 14. Module INPUT (initializing the computation)

Module INPUT is shown in Figure 14. Its functioning is rather clear. All neurons are initially empty, with the exception of neuron σ_{in}, which contains exactly 11 spikes. When a spike enters neuron σ_{in} at time t, the rule $a^2(a^5)^+/a^7 \to a; 1$ is enabled, neurons σ_{d_1}, σ_{d_2}, σ_{d_3}, σ_{d_4} at the next step receive 6 spikes, 2 spikes, 2 spikes, 2 spikes, respectively. After receiving 2 spikes, neuron σ_{d_4} spikes immediately and sends one spike back to neuron σ_{in}, so neuron σ_{in} contains 6 spikes at the next steps until another spike arrives from the environment. Neuron σ_{d_1} contains 6 spikes and does nothing, waiting for the next coming spikes. Neurons σ_{d_2}, σ_{d_3} spike immediately, sending two spikes to each other. This means that at each step both σ_{d_2} and σ_{d_3} fire, hence at each step the content of neuron σ_1 increases by 5 spikes. If at time $t + n$ ($n \geq 1$ is the number to be accepted), the second spike enters neuron σ_{in} coming from the environment, then neuron σ_{in} has 7 spikes and fires again at the next step by rule $a^2(a^5)^+/a^7 \to a; 1$. For neuron σ_{d_1} this entails the firing, and thus neuron σ_{l_0} gets the necessary spikes (2 spikes) to fire;

while for neurons σ_{d_2} and σ_{d_3} this means the end of work, because the spikes are erased by the rule $a^4 \rightarrow \lambda$. Therefore, on the one hand, the content of neuron 1 is $5n$ (it consecutively receives 5 spikes at each step from time $t+2$ to $t+n+2$); on the other hand, neuron σ_{l_0} triggers the simulation of a computation in M, starting with the instruction labeled with l_0. From now on, we start to use modules ADD and SUB associated with register machine M.

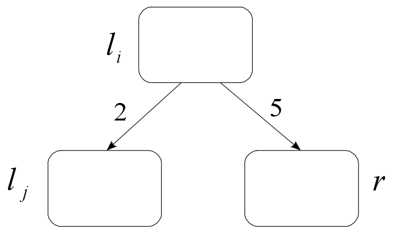

Figure 15. Module ADD in the accepting case

Because ADD instructions in accepting mode are deterministic (of the form $l_i : (\texttt{ADD}(r), l_j)$), the corresponding module ADD is now much simpler than the one in Figure 2, which is shown in Figure 15.

Module SUB remains unchanged, while module FIN is removed, with neuron l_h remaining in the system, but without outgoing synapses. When neuron l_h receives two spikes, it means that the computation of register machine M reaches instruction l_h and stops. Having two spikes inside, neuron l_h fires by rule $a^2 \rightarrow a$, but it sends no spike out because it has no outgoing synapses, and in this way the HSN P system stops. Thus, the computation of the HSN P system stops if and only if the computation in computation in M stops. □

Theorem 5.2. $NRE = N_{acc}HSNP$.

By the way of removing weight on synapses described in Subsection 4.2, we can easily construct modules with usual synapses to replace the modules in Figure 14, 15. Instead of going into details, we just present INPUT module and ADD module in Figures 16 and 17 respectively.

6. Conclusions and Remarks

With a biological motivation, we have considered a restricted variant of spiking neural P systems – homogeneous spiking neural P systems, where all neurons are homogeneous in the sense of having the same set of rules. We have proved that such kind of spiking neural P systems is universal, both when using weighted synapses and using usual synapses (weight on all synapses is 1). This result can have a nice interpretation: the structure of a neural system is crucial for the functioning of the system. Although the individual neuron is simple and homogeneous, by cooperating with each other, a network of neurons can be powerful – “complete (Turing) creativity”.

Corollaries 4.3 and 4.5 show that it is sufficient to have only one neuron having a rule with delay and behaving non-deterministically in order to achieve Turing universality. The role of the unique neuron is essential in the correct simulation of register machine. The non-determinism of a non-deterministic register machine is simulated by the non-deterministic choice of two spiking rules, which is optimal in the number of rules making the system behaves non-deterministically. It remains open whether there is a

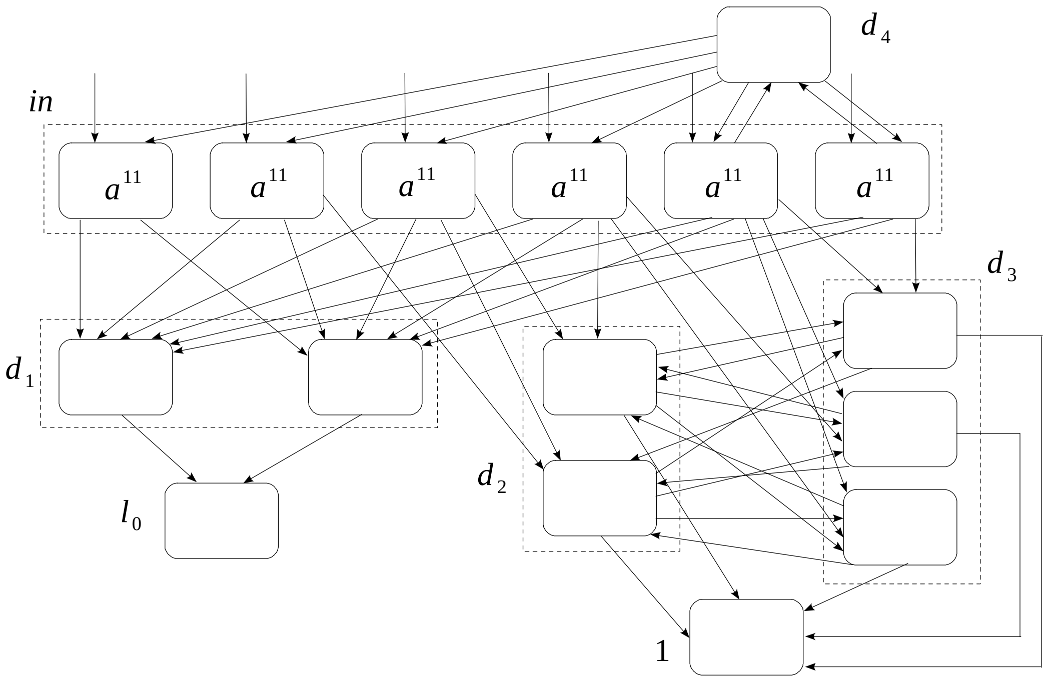

Figure 16. Module INPUT in the accepting mode (without weight on synapses)

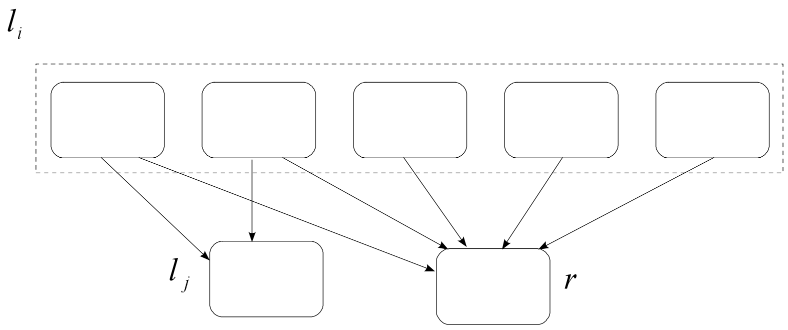

Figure 17. Module ADD in the accepting mode (without weight on synapses)

universal semi-homogeneous spiking neural P system without delay having only one neuron that behaves non-deterministically.

In the proofs of Theorems 4.2 and 5.2, in order to remove weight on synapses, a group of neurons are associated with an instruction of the register machine that we want to simulate. Although the universality of HSN P systems with usual synapses is obtained, the following problem is still of interest. Can we directly construct an HSN P system with usual synapses to simulate a register machine (instead of obtaining a universal HSN P system with usual synapses by modifying a universal HSN P system with weighted synapses), where each instruction is associated with one neuron? Of course, it is expected that in such universal HSN P systems the set of rules is not too complicated.

In Theorems 4.1 and 5.1, the maximal weight on synapses are 5 and 6, respectively. Is it possible to decrease the value of maximal weight on synapses without increasing the number of neurons? (If we can increase the number of neurons, the weight on synapses can be decreased to one as shown in the proofs of Theorems 4.2 and 5.2.)

As usual, it is worth to investigate many other computational properties of HSN P systems under different computational modes and different ways of associating result to computation. An interesting problem among them is to design a universal asynchronous HSN P system.

Acknowledgements

Useful comments by the two 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] Cavaliere, M., Egecioglu, O., Ibarra, O. H., Woodworth, S., Ionescu, M., Păun, Gh.: Asynchronous spiking neural P systems: decidability and undecidability. *Proc. 13th International Meeting on DNA Computing*, (Garzon, M. H., Yan, H. Eds.), LNCS 4848, Springer, Memphis, USA, 2008, 246–255.

[2] Chen, H., Freund, R., Ionescu, M., Păun, Gh., Pérez-Jiménez, M.J.: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, **75**(1-4), 2007, 141–162.

[3] 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*, **7**(2), 2008, 147–166.

[4] Gerstner, W., Kustker,W.: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, Cambridge, 2002.

[5] Ibarra, O.H., Păun, A., Păun, Gh., Rodriguez-Patón, A., Sosik, P., Woodworth, S.: Normal forms for spiking neural P systems. *BWMC2006*, vol. II, 105–136, and *Theoretical Computer Science*, **372**(2-3), 2007, 196–217.

[6] Ibarra, O.H., Woodworth, S.: Characterizations of some restricted spiking neural P systems. *Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers*, LNCS 4361, Springer, 2006, 424–442.

[7] Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. *Fundamenta Informaticae*, **71**(2–3), 2006, 279–308.

[8] Korec, I.: Small universal register machines. *Theoretical Computer Science*, **168**, 1996, 267–301.

[9] Maass, W.: Computing with spikes. Special Issue on Foundations of Information Processing of TELEMATIK, **8**(1), 2002, 32–36.

[10] Maass, W., Bishop, C.: *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.

[11] Minsky, M.: *Computation – Finite and Infinite Machines*. Prentice Hall, 1967.

[12] Pan, L., Păun, Gh., Pérez-Jiménez, M. J.: Spiking neural P systems: a short introduction and new normal forms. *Advanced Computational Technologies* (F. Filip, C. Enachescu, B. Iantovics, eds.), The Pubishing House of the Romanian Academy, Bucharest, 2009.

[13] Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. *Neural Processing Letters,* Submitted.

[14] Păun, A., Păun, Gh.: Small universal spiking neural P systems. *BioSystems*, **90**(1), 2007, 48–60.

[15] Păun, Gh.: *Membrane Computing – An Introduction*. Springer-Verlag, 2002.

[16] Rogozhin, Y.: Small universal Turing machines. *Theoretical Computer Science*, **168**, 1996, 215–240.

[17] Rozenberg, G., Salomaa, A. eds.: *Handbook of Formal Languages, 3 volumes*. Springer-Verlag, Berlin, 1997.

[18] Zhang, X., Zeng X., Pan, L.: Smaller universal spiking neural P systems. *Fundamental Informaticae*, **87**(1), 2008, 117–136.

[19] The P System Web Page: http://ppage.psystems.eu.