ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **17**, Number 1, 2014, 5–18

Small universal sequential spiking neural P systems based on minimum spike number

Keqin JIANG[1,2], Yufang HUANG[2,3], Jinbang XU[2], Zhihua CHEN[2]

[1] School of Computer and Information, Anqing Normal University
Anqing 246133, Anhui, China
E-mail: jiangkq0519@163.com

[2] School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: xujinbang@mail.hust.edu.cn,
chenzhihua@hust.edu.cn (Corresponding author)

[3] College of Mathematics, Southwest Jiaotong University
Chengdu 610031, Sichuan, China
E-mail: huangyufang@home.swjtu.edu.cn

Abstract. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. Recently, a variant of SN P systems was considered: at each step the neuron with the minimum number of spikes among the neurons that can spike will fire. It has been shown, in previous papers, that such systems are Turing complete when the computing result is obtained by accumulating the spikes in the output neuron. In this work, we use a natural way to define the computing result of the systems, by considering the time interval between the first two spikes emitted by the output neuron. As devices for computing functions, we construct a universal sequential SN P system based on minimum spike number (without delay) which uses 137 neurons; as a generator of sets of numbers, a universal sequential SN P system based on minimum spike number (without delay) with 126 neurons is also obtained.

Key-words: membrane computing; spiking neural P system; sequentiality; small universal system

1. Introduction

Spiking neural P systems (SN P systems, for short) were introduced in [5] in the framework of membrane computing as a new class of neural-like P systems which abstracted the way neurons communicate by means of electrical impulses of identical shape, called spikes. Since then, many variants of SN P systems and their properties have been studied [1–4, 6, 10–16, 22–24, 26, 27]; an overview of the field can be found in [18] and [19], with up-to-date information available at the membrane computing website (`http://ppage.psystems.eu`).

Briefly, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph that send *spikes* (electrical impulses) along *synapses* (edges of the graph), under the control of firing rules. One also uses forgetting rules, which remove spikes from neurons. The spikes produced in a neuron are sent to all neurons linked by a synapse to the emitting neuron. The applicability of a rule is usually determined by checking the total number of spikes contained in the neuron against a regular expression associated with the rule. The system works in a synchronized manner, i.e., in each time unit, each neuron which can use a rule should do it (hence the process is parallel at the level of the system and sequential at the level of each neuron).

Recently, a variant of SN P systems was considered in [4]: at each step the neuron with the minimum number of spikes among the neurons that can spike will fire. If there is a tie for the minimum number of spikes stored in the active neurons, only one of the neurons containing the minimum is chosen non-deterministically. We call this strategy "min-sequentiality". It was obtained that SN P systems working in the min-sequentiality manner can achieve Turing completeness when the computing result is obtained by accumulating the spikes in the output neuron.

In computer science, looking for small universal computing devices is a classical research topic, whose aim is to construct computationally universal computing devices using as little as possible resources. Many small universal systems have been developed, such as small universal Turing machines [20], small deterministic Turing machines [8], and small universal register machines [7]. In recent years, looking for small universal SN P systems has been investigated, a research direction initiated in [17], with improved results reported in [28].

Following this line of research, in this work, we consider the problem of looking for small universal min-sequential SN P systems. Here, we use the standard way to define the computing result of an SN P system as the time interval between the first two spikes emitted by the output neuron. Two versions of min-sequential SN P systems are considered, that is, as devices computing functions and as devices generating sets of numbers. As devices for computing functions, we construct a universal min-sequential SN P system (without delay) which uses 137 neurons; as a generator of sets of numbers, a universal min-sequential SN P system (without delay) with 126 neurons is also obtained.

2. Min-Sequential Spiking Neural P Systems

We start by recalling the definition of min-sequential SN P systems (the feature of delays is omitted, because it is not used in this work). In the definition of these systems, the notion of regular expression is used; please refer to [21] for the details.

A min-sequential SN P system, of degree $m \geq 1$, is a construct of the form

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

- $O = \{a\}$ is a singleton alphabet (a is called *spike*);
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$ with $1 \leq i \leq m$, where
  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) $E/a^c \to a$, where E is a regular expression over O and $c \geq 1$;
     (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \to a$ from R_i;
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
- $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

The rules of the form $E/a^c \to a$ are *firing* (we also say *spiking*) rules, which are applied as follows. If neuron σ_i contains k spikes, and $a^k \in L(E)$, $k \geq c$, then the rule $E/a^c \to a \in R_i$ can be used. This means c spikes are consumed (thus only $k - c$ spikes remain in neuron σ_i), and one spike is sent out to all neurons σ_j such that $(i, j) \in syn$. The rules of the form $a^s \to \lambda$ are *forgetting* rules. If neuron σ_i contains exactly s spikes, then the forgetting rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

We note that the rules are used in the sequential manner in each neuron, but in previous literature neurons function in parallel with each other. In the following we will consider another strategy for rule application: at each step, the neuron with the minimum number of spikes among the neurons that are active (can spike) will fire. If there is a tie for the minimum number of spikes stored in the active neurons, only one of the neurons containing the minimum is chosen non-deterministically. This strategy makes the system sequential; we will call this "min-sequentiality".

The *configuration* of the system is described by the numbers $n_1, \ldots, n_m$ of spikes present in each neuron. It is denoted by $\langle n_1, \ldots, n_m \rangle$. Using the rules as described above, we can define *transitions* from one configuration to another. A series of transitions starting from the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used.

Similar with the case considered in [17], in order to compute a function $f : N^k \to N$, we need to introduce k natural numbers $n_1, \ldots, n_k$ in the system by "reading" from the environment a binary sequence $z = 10^{2n_1}10^{2n_2}1 \ldots 10^{2n_k}1$. This means that

the input neuron of the system receives a spike in each step corresponding to a digit 1 from the string z and no spike otherwise. Note that we input exactly $k+1$ spikes, i.e., after the last spike we assume that no further spike is coming to the input neuron. The result of the computation is encoded in the distance between the first two spikes emitted by the output neuron. If the system never halts, then no result is considered. Specifically, the system produces a spike train of the form $0^b10^{4r-2}1$, for some $b \geq 0$, then we have $r = f(n_1, \ldots, n_k)$. Note that the spike train produced in this work is different with the classical one as given in [17].

3. Small Universal Min-Sequential SN P Systems (Computing approach)

In this section, we attempt to construct a small universal min-sequential SN P system used for computing functions.

The following proof is based on the simulation of deterministic register machines given in the form $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H is associated with only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j)$ (add 1 to register r, then go to the instruction with label l_j),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A deterministic register machine can work in the accepting mode: a number n is introduced in the first register (all other registers are empty) and we start to apply the instruction with label l_0; if the computation eventually halts, then the number n is accepted.

A deterministic register machine can also compute any Turing computable function (we assume that the function to be computed has arguments $n_1, \ldots, n_k$): 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 instruction with label l_0, 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 $M(n_1, n_2, \ldots, n_k)$.

Theorem 1. *There exists a universal computing min-sequential SN P system (without delay) having 137 neurons, where the result is encoded in the distance of the first two spikes emitted by the output neuron.*

Proof. A register machine can compute any Turing computable function (see e.g., [9]). We will show that a universal register machine from [7] (shown in Fig. 1) can be

simulated by an SN P system working in min-sequentiality manner. Let $(\varphi_0, \varphi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y we have $\varphi_x(y) = M_u(g(x), y)$. As in the case considered in [17], we further add a register 8, which is never decremented during the computation, and we replace the old halt instruction of M_u with the following three instructions: $l_h : (\texttt{SUB}(0), l_{22}, l'_h), l_{22} : (\texttt{ADD}(8), l_h)$, and $l'_h : \texttt{HALT}$. Thus at the end of the program of the initial machine we will copy the contents of the old output register (register 0) into the register 8 which will serve as the new output register. We do this so that no substraction rule will be applicable to the output register. In this way, we have 9 registers (numbered from 0 to 8), 24 ADD and SUB instructions, and 25 labels. We denote by M'_u the obtained register machine.

$l_0 : (\texttt{SUB}(1), l_1, l_2)$, $l_1 : (\texttt{ADD}(7), l_0)$,
$l_2 : (\texttt{ADD}(6), l_3)$, $l_3 : (\texttt{SUB}(5), l_2, l_4)$,
$l_4 : (\texttt{SUB}(6), l_5, l_3)$, $l_5 : (\texttt{ADD}(5), l_6)$,
$l_6 : (\texttt{SUB}(7), l_7, l_8)$, $l_7 : (\texttt{ADD}(1), l_4)$,
$l_8 : (\texttt{SUB}(6), l_9, l_0)$, $l_9 : (\texttt{ADD}(6), l_{10})$,
$l_{10} : (\texttt{SUB}(4), l_0, l_{11})$, $l_{11} : (\texttt{SUB}(5), l_{12}, l_{13})$,
$l_{12} : (\texttt{SUB}(5), l_{14}, l_{15})$, $l_{13} : (\texttt{SUB}(2), l_{18}, l_{19})$,
$l_{14} : (\texttt{SUB}(5), l_{16}, l_{17})$, $l_{15} : (\texttt{SUB}(3), l_{18}, l_{20})$,
$l_{16} : (\texttt{ADD}(4), l_{11})$, $l_{17} : (\texttt{ADD}(2), l_{21})$,
$l_{18} : (\texttt{SUB}(4), l_0, l_h)$, $l_{19} : (\texttt{SUB}(0), l_0, l_{18})$,
$l_{20} : (\texttt{ADD}(0), l_0)$, $l_{21} : (\texttt{ADD}(3), l_{18})$,
$l_h : \texttt{HALT}$

Fig. 1. A small universal register machine from Korec [7].

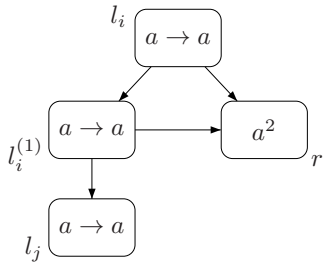

Fig. 2. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j)$).

In the SN P system we construct, neurons are associated with each register and with each label of an instruction of the machine. Specifically, we start with two spikes in the neurons modeling the registers. If a register r contains a number n, then the associated neuron will contain $2n + 2$ spikes. The simulations of ADD and SUB instructions are done by the modules presented in Figs. 2 and 3, which are different from those in [4].

The ADD module works as follow: when one spike enters neuron σ_{l_i}, this neuron fires sending one spike to each of neurons $\sigma_{l_i^{(1)}}, \sigma_r$, thus neurons $\sigma_{l_i^{(1)}}$ and σ_r contain 1 spike and $2n+3$ ($n \geq 0$) spikes, respectively, both of them are activated (neuron σ_r will be activated due to the applicability of the rule $a(a^2)^+/a^3 \to a$ or $a(a^2)^+/a^2 \to a$). Due to the min-sequentiality, neuron $\sigma_{l_i^{(1)}}$ fires at the next step, sending one spike to neurons σ_{l_j} and σ_r, respectively. In this way, neuron σ_{l_j} is the only active one and it fires, and the number of spikes in neuron σ_r is increased by two, which simulates the increase of the number stored in register r by 1.

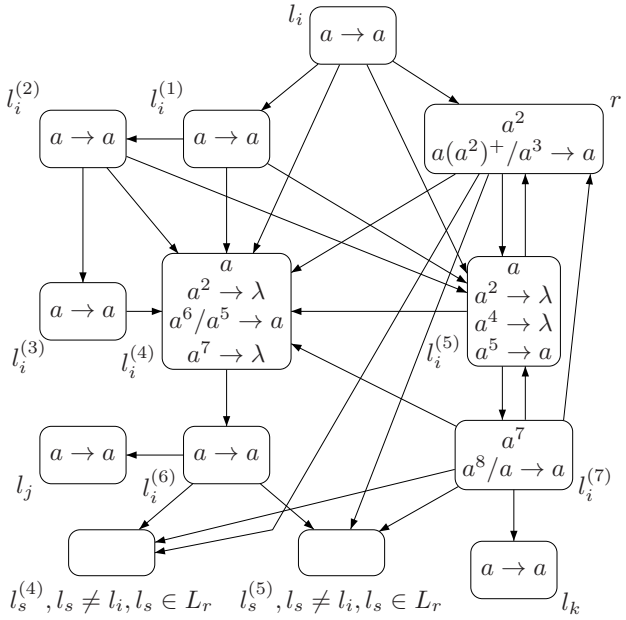

Fig. 3. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$), where $L_r = \{l \mid l \text{ is a label of a SUB instruction acting on the register } r\}$.

The simulation of a SUB instruction is shown in Fig. 3. When neuron σ_{l_i} fires at step t, neurons $\sigma_{l_i^{(1)}}, \sigma_{l_i^{(4)}}, \sigma_{l_i^{(5)}}$ and σ_r receive one spike, respectively; so neuron σ_r contains $2n + 3$ ($n \geq 0$) spikes (corresponding to the fact that the number stored in register r is n). Due to the min-sequentiality, neuron $\sigma_{l_i^{(1)}}$ fires at step $t+1$ sending a spike to each of neurons $\sigma_{l_i^{(2)}}, \sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$; thus neurons $\sigma_{l_i^{(2)}}, \sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$ contain 1, 3, 3 spikes, respectively. At step $t+2$, neuron $\sigma_{l_i^{(2)}}$ fires sending one spike to each of

neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$. At step $t+3$, neuron $\sigma_{l_i^{(3)}}$ fires sending a spike to neuron $\sigma_{l_i^{(4)}}$, thus at the next step neurons $\sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$ contain 5 and 4 spikes, respectively, whereas neuron σ_r contains $2n+3 (n \geq 0)$ spikes. Two possible cases follow:

(1) At step $t+4$, the number of spikes in neuron σ_r is 3 (corresponding to the fact that the number stored in register r is 0). In this case, neuron σ_r fires since it has three spikes (one less than neuron $\sigma_{l_i^{(5)}}$), and sends a spike to each of neurons $\sigma_{l_i^{(4)}}, \sigma_{l_i^{(5)}}$. Let L_r be the set of labels of SUB instructions acting on register r. Neuron σ_r has synapses with all the neurons $\sigma_{l_s^{(4)}}$ and $\sigma_{l_s^{(5)}}$, $l_s \in L_r$, which means all of them receive a spike from σ_r. In this way, neuron $\sigma_{l_i^{(4)}}$ contains 6 spikes, neuron $\sigma_{l_i^{(5)}}$ contains 5 spikes, each of neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \neq l_i)$ contains 2 spikes, hence all of these neurons are active. Due to the min-sequentiality strategy, neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \neq l_i)$ forget their spikes using the forgetting rule $a^2 \to \lambda$ (at each step, non-deterministically chosen, one of neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \neq l_i)$ fires). After all of neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \neq l_i)$ forget their spikes, neuron $\sigma_{l_i^{(5)}}$ fires by the rule $a^5 \to a$ sending a spike to neurons $\sigma_{l_i^{(4)}}, \sigma_{l_i^{(7)}}$ and sending another one back to neuron σ_r. At the next step, neurons $\sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(7)}}$ are activated (holding 7 spikes and 8 spikes), so neuron $\sigma_{l_i^{(4)}}$ removes its seven spikes by the rule $a^7 \to \lambda$. At the next step, neuron $\sigma_{l_i^{(7)}}$ fires sending a spike to each of neurons σ_{l_k}, σ_r, $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ for any $l_s \in L_r$. In this way, the numbers of spikes in neurons σ_r and $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \in L_r)$ are reset to the value they had at the beginning of this simulation, which ensures that another SUB instruction can be correctly simulated at a subsequent step. When neuron σ_{l_k} fires, system Π starts to simulate the instruction l_k of M.

(2) At step $t+4$, neuron σ_r has $2n+3$ $(n \geq 1)$ spikes, both of neurons $\sigma_{l_i^{(5)}}$ and σ_r are active. Due to the min-sequentiality, neuron $\sigma_{l_i^{(5)}}$ removes its four spikes by the forgetting rule $a^4 \to \lambda$. At step $t+5$, σ_r is the unique active neuron, and it fires sending one spike to each of neurons $\sigma_{l_s^{(4)}}$ and $\sigma_{l_s^{(5)}}$, where $l_s \in L_r$, $L_r = \{l \mid l$ is a label of a SUB instruction acting on the register $r\}$. In this way, neuron σ_r has $2(n-1)+2$ spikes, which simulates that the number stored in register r is decreased by one; neuron $\sigma_{l_i^{(5)}}$ contains one spike; neuron $\sigma_{l_i^{(4)}}$ contains six spikes; each of neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \in L_r,\ l_s \neq l_i)$ contains 2 spikes. Due to the min-sequentiality strategy, neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \in L_r,\ l_s \neq l_i)$ forget their spikes by the rule $a^2 \to \lambda$. After all of neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \neq l_i)$ forget their spikes, neuron $\sigma_{l_i^{(4)}}$ fires by the rule $a^6/a^5 \to a$ sending a spike to neuron $\sigma_{l_i^{(6)}}$ (one spike remains in $\sigma_{l_i^{(4)}}$). Note that, at this moment, the numbers of spikes in neurons $\sigma_{l_i^{(4)}}$ and $\sigma_{l_i^{(5)}}$ are reset to the value they had at the beginning of this simulation. At the next step, neuron $\sigma_{l_i^{(6)}}$ fires sending a spike to each of neurons σ_{l_j}, $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \in L_r,\ l_s \neq l_i)$. So, the numbers of spikes in neurons $\sigma_{l_s^{(4)}}, \sigma_{l_s^{(5)}}$ $(l_s \in L_r,\ l_s \neq l_i)$ are also reset to the number of

spikes they had at the beginning of this simulation; neuron σ_{l_j} has one spike. When neuron σ_{l_j} fires, system Π starts to simulate the instruction l_j of M.

In order to simulate the universal register machine M'_u (it has 9 registers, numbered from 0 to 8), we begin with neurons σ_1 and σ_2 already loaded with $2g(x) + 2$ and $2y+2$ spikes, respectively. The work of the system is triggered by introducing one spike in the neuron σ_{l_0} (associated with the starting instruction of the register machine). Then, the system goes into the phase of simulating the instructions of the universal register machine M'_u by the modules from Figs. 2 and 3. In this way, this system can work in the same way as the universal register machine M'_u. Hence, neuron σ_8 will contain $2\varphi_x(y) + 2$ spikes when the computation halts.

Two problems remain to be solved for simulating the small universal register machine M'_u, namely, introducing the mentioned spikes in neurons σ_{l_0}, σ_1, σ_2, and outputting the computed number. The first problem is solved by the module INPUT presented in Fig. 4. If we "read" a spike train $10^{2g(x)}10^{2y}1$ from the environment, then the module works as follows: the first spike from neuron σ_{in} is sent to each of neurons σ_{c_1}, σ_{c_3} and σ_{c_5}. Thus, neuron σ_{c_1} fires sending a spike to each of neurons σ_{c_2} and σ_1. Neurons σ_{c_2} and σ_1 are activated (contain 2 spikes and 3 spikes). Due to the min-sequentiality, neuron σ_{c_2} fires sending a spike to neurons σ_{c_1}, σ_1, respectively. In this way, neuron σ_{c_1} again has 3 spikes, which means that neurons σ_{c_1} and σ_{c_2} will fire alternatively, as long as no new spike is received in σ_{in}, and the number of spikes in neuron σ_1 is increased by two after each iteration. When the second spike enters neuron σ_{in}, this neuron fires first because it has the least number of spikes in the system. After receiving one spike from σ_{in}, neuron σ_{c_1} is blocked. However, neuron σ_{c_3} is activated and will push the desired spikes into neuron σ_2 in the same way. Neuron σ_{c_5} fires only after receiving the third spike from neuron σ_{in}, and then it sends a spike to neuron σ_{l_0}, thus the simulation of M'_u is triggered.

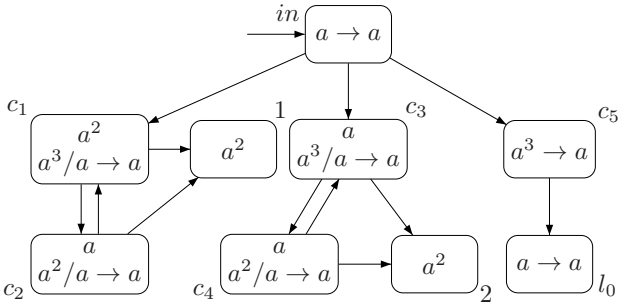

Fig. 4. Module INPUT.

We now consider the second problem, that is, outputting the computed number. As the usual way of encoding information used in SN P systems, the computing result is encoded in a specific form of the time interval between the first two spikes emitted by the output neuron. If the system produces a spike train of the form $0^b10^{4r-2}1$, for some $b \geq 0$, then it means that the produced number is r (neuron σ_8 contains $2r + 2$ spikes). The OUTPUT module is shown in Fig. 5 and it works in the following way.

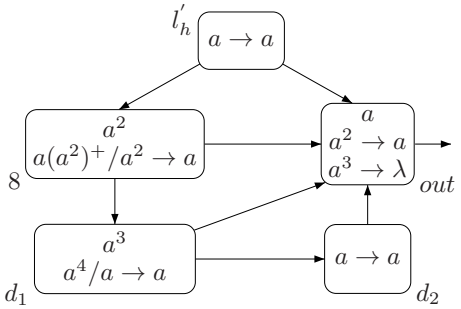

Fig. 5. Module OUTPUT.

When neuron $\sigma_{l'_h}$ receives one spike, it fires and sends a spike to neurons σ_8 and σ_{out}. Thus, the two neurons are activated. Due to the min-sequentiality, neuron σ_{out} fires by the rule $a^2 \to a$ and the first spike of the spike train is produced. At the next step, neuron σ_8 fires sending a spike to each of neurons σ_{d_1} and σ_{out}; thus at the next step neuron σ_{d_1} fires sending a spike to neurons σ_{d_2} and σ_{out}. At the next step, neurons $\sigma_{d_2}, \sigma_{out}$ and σ_8 are all activated, neuron σ_{d_2} fires sending a spike to neuron σ_{out}. At the next step, neuron σ_{out} removes its spikes by the rule $a^3 \to \lambda$. At the next step, neuron σ_8 fires, the computation returns to a configuration similar to the initial one, so this process can continue. When neuron σ_8 goes down to only having three spikes (one less than neuron σ_{d_1} at that step), then it fires again (thus, fires at two consecutive steps). After receiving one spike from neuron σ_8, neuron σ_{d_1} is blocked, and neuron σ_{out} fires, the second spike of the spike train is produced.

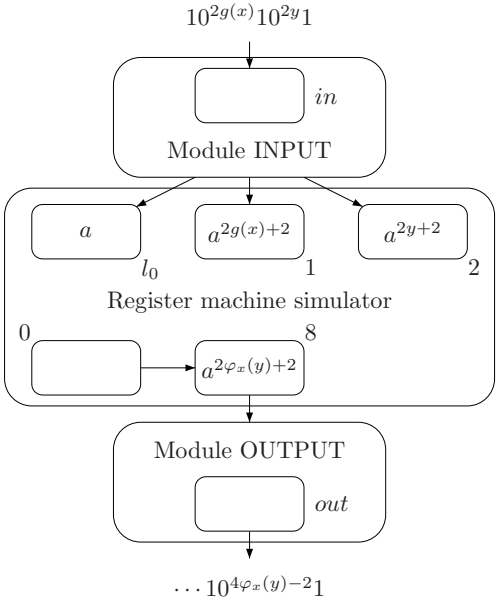

Fig. 6. The general design of the universal SN P system.

From the above description of the modules and their work, it is clear that the register machine is correctly simulated by the system in the min-sequentiality manner. The overall design of the small universal computing system is given in Fig. 6.

We can easily check that we need 9 neurons for the 9 registers, 25 neurons for the 25 labels, 98 neurons for the 14 SUB instructions, 10 neurons for the 10 ADD instructions, 6 neurons in the INPUT module, 3 neurons in the OUTPUT module. Therefore, the total number of neurons is 151.

This number can be slightly decreased, by some "code optimization", exploiting some particularities of the register machine M'_u.

For the sequence of two consecutive ADD instructions $l_{17} : (\texttt{ADD}(2), l_{21})$ and $l_{21} : (\texttt{ADD}(3), l_{18})$ without any other instruction addressing the label l_{21}, we can combine the two ADD modules to simulate this sequence by the module in Fig. 7, where they share the auxiliary neuron $\sigma_{l_{17}^{(1)}}$. In this way we save 2 neurons in total.

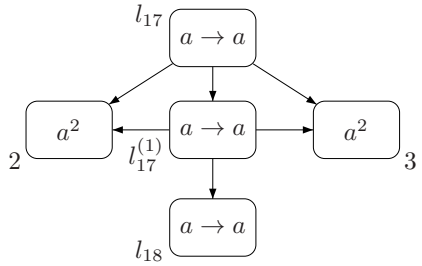

Fig. 7. A module simulating two consecutive ADD instructions.

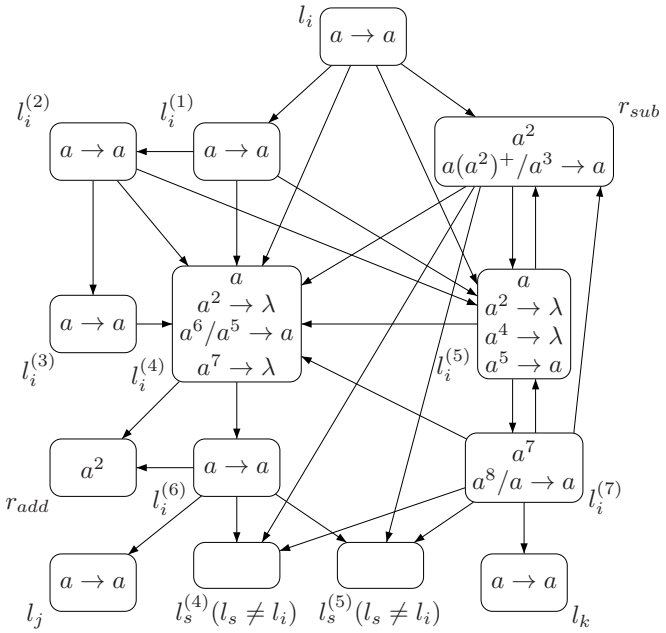

Fig. 8. A module simulating consecutive SUB-ADD instructions.

A similar operation is possible for the following six sequences of SUB-ADD instructions, where we can save the intermediate labels $(l_1, l_5, l_7, l_9, l_{16}, l_{22})$, as well as one auxiliary neuron for each pair:

$l_0 : (\texttt{SUB}(1), l_1, l_2)$, $l_1 : (\texttt{ADD}(7), l_0)$,
$l_4 : (\texttt{SUB}(6), l_5, l_3)$, $l_5 : (\texttt{ADD}(5), l_6)$,
$l_6 : (\texttt{SUB}(7), l_7, l_8)$, $l_7 : (\texttt{ADD}(1), l_4)$,
$l_8 : (\texttt{SUB}(6), l_9, l_0)$, $l_9 : (\texttt{ADD}(6), l_{10})$,
$l_{14} : (\texttt{SUB}(5), l_{16}, l_{17})$, $l_{16} : (\texttt{ADD}(4), l_{11})$,
$l_h : (\texttt{SUB}(0), l_{22}, l'_h)$, $l_{22} : (\texttt{ADD}(8), l_h)$.

Instead of modules ADD and SUB as in Figs. 2 and 3, for each couple of instructions as above we can use a module as shown in Fig. 8. In each case, we save two neurons; together with the two neurons saved by the module in Fig. 7, an improvement is achieved from 151 to 137 neurons. This completes the proof. □

4. Small Universal Min-Sequential SN P Systems (Generative approach)

Let us now consider the case of small universal min-sequential SN P systems as number generators.

As stated in [17], a generating SN P system Π_u is universal if, given a fixed admissible enumeration of the unary partial recursive functions, $(\varphi_0, \varphi_1, \ldots)$, there is a recursive function g such that for each natural number x, if we input the number $g(x)$ in Π_u, the set of numbers generated by the system is equal to $\{n \in N | \varphi_x(n) \text{ is defined}\}$. Otherwise stated, after introducing the "code" $g(x)$ of the partial recursive function φ_x in a specified neuron, the system generates all numbers n for which $\varphi_x(n)$ is defined. However, it is necessary to mention that we introduce here the number $g(x)$ by "reading" the sequence $10^{2g(x)}1$ from the environment instead of the sequence $10^{g(x)-1}1$ which was used in [17].

The strategy followed by the universal system Π_u is the following:

(1) Read the string $10^{2g(x)}1$ from the environment and load $2g(x)$ spikes in neuron σ_1.

(2) Load neuron σ_2 non-deterministically with an arbitrary natural number n (introducing $2n$ spikes in neuron σ_2); at the same time, output the spike train $10^{4n-2}1$ (hence the number n).

(3) Check whether the function φ_x is defined for n. To this aim, start the register machine M_u from Fig. **??**, with $g(x)$ in register 1 and n in register 2. If the computation in M_u halts, then also the computation in our sequential SN P system halts, hence n is introduced in the set of generated numbers.

Note that there is an essential difference between number generating and function computing. In the case of sequential SN P systems as number generators, we no longer need to output a result after halting the computation, but we have to randomly

generate a number at the beginning of the computation. Hence, we need a few modifications to the construction of the universal computing min-sequential SN P systems. First, no separate OUTPUT module is necessary, the additional register 8 can be omitted, and the label l_h can be saved by just letting the computation halt. Second, the INPUT module should be combined with the output one, at the same time non-deterministically producing the number n.

The combined INPUT-OUTPUT module is presented in Fig. 9.

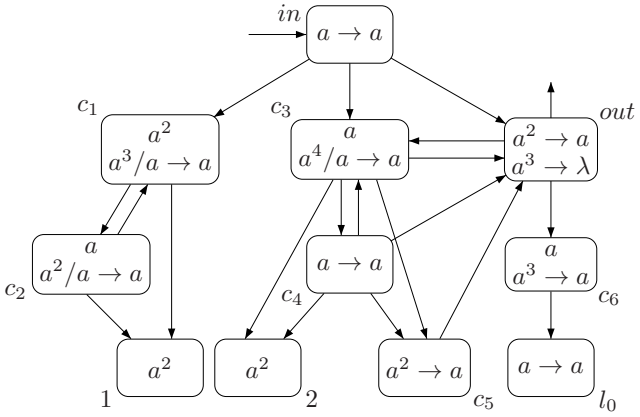

Fig. 9. The INPUT–OUTPUT module for number generating universal systems.

After loading neuron σ_1 with $2g(x)$ spikes, we start loading neuron σ_2 with an arbitrary number of spikes by means of neurons σ_{c_3} and σ_{c_4}, at the same time sending out the respective number as the time interval between the first two spikes emitted by neuron σ_{out}. Note that when receiving the spikes from neuron σ_{c_4}, both σ_{c_5} and σ_{out} are activated, both have exactly two spikes at this moment, so we need to choose non-deterministically between them the one to fire next. As long as neuron σ_{c_5} fires, then neuron σ_{out} forgets its three spikes by the rule $a^3 \to \lambda$, the arbitrary number will be generated in neuron σ_2. The work of neurons σ_{c_3} and σ_{c_4} stops when neuron σ_{out} uses the rule $a^2 \to a$, and only after that (after having two spikes emitted by σ_{out}) we load neuron σ_{l_0} with one spike (via neuron σ_{c_6}). In this way, the work of the register machine M_u is triggered.

Consequently, the obtained system contains: 8 neurons for the 8 registers, 22 neurons for the 22 labels (l_h is saved), 91 neurons for the 13 SUB instructions, 9 neurons for the 9 ADD instructions, 8 neurons in the INPUT-OUTPUT module. The total number of neurons is 138. From the observation in Section 3, we can save 12 neurons (five sequences of SUB-ADD instructions and one ADD-ADD instruction). Hence, we have the following result:

Theorem 2. *There is a universal number generating min-sequential SN P system (without delay) having 126 neurons, where the result is encoded in the distance of the first two spikes emitted by the output neuron.*

5. Conclusions and Remarks

In this work, the problem of constructing a small universal sequential spiking neural P system based on minimum spike number is addressed when the computing result is encoded in a specific form of the time interval between the first two spikes emitted by the output neuron. As devices for computing functions, we construct a universal min-sequential SN P system (without delay) which uses 137 neurons; as a generator of sets of numbers, a universal min-sequential SN P system (without delay) with 126 neurons is also obtained.

In [25], a new way is introduced for simulating register machines by spiking neural P systems, where only one neuron is used for each instruction of the register machine; in this way, less neurons can be used to construct universal spiking neural P systems. It is interesting whether we can construct a universal sequential SN P system based on minimum spike number with less neurons following this proof technique.

Acknowledgements. This work was supported by National Natural Science Foundation of China (61370105, 61033003, 91130034 and 61320106005), Fundamental Research Funds for the Central Universities (2013TS124), Ph.D. Programs Foundation of Ministry of Education of China (20120142130008), Natural Science Foundation of Hubei Province (2013CFB159), Anhui Provincial Natural Science Foundation (1408085MF131), Natural Science Research Project for Higher Education Institutions of Anhui Province(KJ2014A140, KJ2013B119).

References

[1] CAVALIERE M., IBARRA O.H., PĂUN G., EGECIOGLU O., IONESCU M., WOODWORTH S., *Asynchronous spiking neural P systems*, Theoretical Computer Science, 2009, Vol. **410**(24), pp. 2352–2364.

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

[3] IBARRA O.H., PĂUN A., PĂUN G., RODRÍGUEZ-PATÓN A., SOSÍK P., WOODWORTH S., *Normal forms for spiking neural P systems*, Theoretical Computer Science, 2007, Vol. **372**, pp. 196–217.

[4] IBARRA O.H., PĂUN A., RODRÍGUEZ-PATÓN A., *Sequential SNP systems based on min/max spike number*, Theoretical Computer Science, 2009, Vol. **410**, pp. 2982–2991.

[5] IONESCU M., PĂUN G., YOKOMORI T., *Spiking neural P systems*, Fundamenta Informaticae, 2006, Vol. **71**(2–3), pp. 279–308.

[6] ISHDORJ T.-O., LEPORATI A., PAN L., ZENG X., ZHANG X., *Deterministic solutions to* QSAT *and* Q3SAT *by spiking neural P systems with pre-computed resources*, Theoretical Computer Science, 2010, Vol. **411**(25), pp. 2345–2358.

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

[8] KUDLEK M., *Small deterministic Turing machines*, Theoretical Computer Science, 1996, Vol. **168**(2), pp. 241–255.

[9] MINSKY M., *Computation – Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[10] PAN L., PĂUN G., *Spiking neural P systems with anti-spikes*, International Journal of Computers, Communications & Control, 2009, Vol. **3**, pp. 273–282.

[11] PAN L., PĂUN G., *Spiking neural P systems: An improved normal form*, Theoretical Computer Science, 2010, Vol. **411**(6), pp. 906–918.

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

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

[14] PAN L., ZENG X., *Small universal spiking neural P systems working in exhaustive mode*, IEEE Transactions on Nanobioscience, 2011, Vol. **10**(2), pp. 99–105.

[15] PAN L., ZENG X., ZHANG X., *Time-free spiking neural P systems*, Neural Computation, 2011, Vol. **23**, pp. 1–23.

[16] PAN L., ZENG X., ZHANG X., JIANG Y., *Spiking neural P systems with weighted synapses*, Neural Processing Letters, 2012, Vol. **35**(1), pp. 13–27.

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

[18] PĂUN Gh., *Membrane Computing – An Introduction*, Springer-Verlag, Berlin, 2002.

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

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

[21] ROZENBERG G., SALOMAA A. (eds.), *Handbook of Formal Languages, 3 volumes*, Springer-Verlag, Berlin, 1997.

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

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

[24] WANG J., HOOGEBOOM H.J., PAN L., PĂUN Gh., PÉREZ-JIMÉNEZ M.J., *Spiking neural P systems with weights*, Neural Computation, 2010, Vol. **22**(10), pp. 2615–2646.

[25] ZENG X., LU C., PAN L., *A weakly universal spiking neural P system*, Mathematical and Computer Modelling, 2010, Vol. **52**, pp. 1940–1946.

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

[27] ZHANG X., ZENG X., PAN L., *On languages generated by asynchronous spiking neural P systems*, Theoretical Computer Science, 2009, Vol. **410**, pp. 2478–2488.

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