# A normal form of spiking neural P systems with structural plasticity

Tao Song and Linqiang Pan*

Key Laboratory of Image Information Processing
and Intelligent Control,
School of Automation,
Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
Email: taosong@hust.edu.cn
Email: lqpan@hust.edu.cn
*Corresponding author

**Abstract:** Spiking neural P systems (SN P systems, in short) are a class of distributed and parallel neural-like computing models, which are inspired by the way of biological neurons processing information and communication by means of impulses (or spikes). SN P systems with structural plasticity are a new variant of SN P systems, which have plasticity to the topological structure in the sense that connections among neurons can be reformed by creating and deleting synapses during the computation. In this work, we give a normal form of universal SN P systems with structural plasticity. Specifically, we prove that SN P systems with structural plasticity can achieve Turing universality with the restriction that any neuron can either create or delete synapses, instead of both, in a computation step. This result gives an answer to an open problem formulated in Francis et al. (2013).

**Keywords:** bio-inspired computing; membrane computing; spiking neural P system; SN P system; Turing universality; structural plasticity.

**Reference** to this paper should be made as follows: Song, T. and Pan, L. (2015) 'A normal form of spiking neural P systems with structural plasticity', *Int. J. Swarm Intelligence*, Vol. 1, No. 4, pp.344–357.

**Biographical notes:** Tao Song received his PhD from Huazhong University of Science and Technology, China in 2013. He was granted a post-doctor in Huazhong University of Science and Technology. His research interests include membrane computing and spiking neural P systems.

Linqiang Pan received his PhD from Nanjing University, China in 2000. Since 2004, he has been a Professor at Huazhong University of Science and Technology. His research interests include membrane computing, systems biology, and graph theory.

## 1 Introduction

*Membrane computing* is a branch of bio-inspired computing, which was initiated by Păun (2000). The aim of membrane computing is to construct distributed and parallel computing models/devices as well as design intelligent algorithms by abstracting ideas (such as data structures, ways to store and process information and communication strategies) from the structure and the functioning of single cell or from tissues and organs, such as the brain. The obtained computing models are usually called *P systems*. There are three classes of mainly investigated P systems: cell-like P systems (based on a cell-like (hence hierarchical) arrangement of membranes delimiting compartments where multisets of chemicals evolve according to given evolution rules) (Păun, 2000), tissue-like P systems (instead of hierarchical arrangement of membranes, consider arbitrary graphs as underlying structures, with membranes placed in the nodes while edges correspond to communication channels) (Martín-Vide et al., 2003) and neural-like P systems. An overview of the field can be found in Păun (2002) and Păun et al. (2010). The present work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short), introduced in Ionescu et al. (2006).

SN P systems were abstracted from the way the neurons process information and communicate with each other by sending spikes along synapses. An SN P system can be represented by a directed graph, where neurons are placed in the nodes and each neuron sends spikes to its neighbour neurons along the synapses (represented by the arcs of the graph). Each neuron can contain a number of copies of a single object type, called the *spike*, several *spiking* rules and *forgetting* rules. Using its rules, a neuron can send information (in the form of spikes) to other neurons. One of the neurons is designated as the output neuron and its spikes are also sent to the environment. When a spike is emitted by the output neuron, we marked that time with 1; otherwise, we marked it with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop. A result can be associated with a computation in various ways: for example, as the number of spikes sent to the environment (in the case of halting computations) or as the time elapsed between the first two consecutive spikes sent to the environment by the (output neuron of the) system.

SN P systems can be used as computing devices mainly in the following three ways: generating (computing) sets of numbers (Ionescu et al., 2006; Ibarra et al., 2007; Pan and Păun, 2010), generating languages (Chen et al., 2007; Zhang et al., 2008a) and computing recursive functions (Pan and Zeng, 2010; Păun and Păun, 2007; Zhang et al., 2008b). Moreover, SN P systems with neuron budding and cell division can (theoretically) generate exponential working space in linear time, thus can provide a way to solve computationally hard problems in a feasible (polynomial or linear) time (see, e.g., Leporati et al., 2009; Pan et al., 2011). Some applications of SN P systems, particular in simulating logic gates and circuits, were also developed (Ionescu and Sburlan, 2012; Zhang et al., 2012). Inspired by different biological facts, many variants of SN P systems have also been investigated, such as asynchronous SN P systems (Cavaliere et al., 2009), asynchronous SN P systems with local synchronisation (Song et al., 2013a), reversible SN P systems (Song et al., 2013b), SN P systems with astrocyte-like control (Pan et al., 2012; Macías-Ramos and Pérez-Jiménez, 2013; Păun, 2007), SN P systems with rules on synapses (Song et al., 2014a), SN P systems with anti-spikes (Pan and Păun, 2009; Song et al., 2014b; Metta and Kelemenová, 2014), SN P systems with inhibitory synapse (Song and Wang, 2014). Most of the variants

of SN P systems have been proved to be Turing universal, where various biological features are abstracted as operations to make the systems perform computation in a similar way as that of living entities doing computation. SN P systems provide a rich family of neural-like computing systems with powerful computability, and may have potential applications in solving realistic problems.

The present work deals with a variant of SN P systems, called *SN P systems with structural plasticity*, The systems are inspired from the biological neural plasticity of synapse, i.e., topology modification by means of synapse creation, deletion and rewiring (Francis et al., 2013). Structural plasticity is concerned with any change in connectivity between neurons with two mechanisms:

1 synaptogenesis and synapse deletion

2 synaptic rewiring.

So, in SN P systems with structural plasticity, besides spiking and forgetting rules, a neuron can also contain plasticity rules, by which a neuron can create and delete connections (synapses) to its neighbouring neurons. It was shown that SN P systems with structural plasticity are able to compute all Turing computable sets of numbers by using standard rules (a spiking rule can produce one spike in each time of spiking), where a neuron is allowed to both create and delete the connections to certain neighbour neurons in a computation step (Francis et al., 2013). It was formulated as an open problem whether we can achieve universality with the restriction that a neuron is allowed to create or delete, instead of both, connections to some of its neighbour neurons in a computation step.

In this work, we give an answer to this open problem. Specifically, we prove that SN P systems with structural plasticity can compute any set of Turing computable natural numbers, with the restriction that each neuron is allowed to either create connection or delete connection to certain neighbouring neurons in a computation step. Note that the universality of SN P systems with structural plasticity in Francis et al. (2013) was obtained by simulating register machines in a constructive way, where some modules (ADD and SUB modules corresponding to instructions of a register machine) were constructed. In our proof, the result is achieved by simulating register machines as used in Francis et al. (2013), but the considered modules can avoid undesired interfere and perform computation in a 'safe' way.

The SN P system with structural plasticity has a flexible topological structure, that is, particular neurons can make connection only when it contains specific number of spikes. This is a quite different feature comparing with universal SN P systems obtained in Chen et al. (2007), Zhang et al. (2008a), Song et al. (2013b), Pan et al. (2012), Macías-Ramos and Pérez-Jiménez (2013) and Păun (2007). The topological structures of those systems are fixed as initially constructed and cannot change during the computation. The feature of delay plays a vital role in the universality proofs of classic SN P systems, see e.g., Ionescu et al. (2006), while in universality proof of SN P system with structural plasticity, the feature of delay is not used with the applications of synaptogenesis and synapse deletion. In Ibarra et al. (2007), SN P systems without delay have been proved to universal, but the modules were complex. For instance, the ADD module in Ibarra et al. (2007) has 13 neurons and 18 initial synapses, while in the ADD module constructed for universal SN P systems structural plasticity, there are eight neurons and nine initial synapses (see below in details). These results may give

some hints to construct universal SN P systems having flexible and simple topological structure by means of synaptogenesis and synapse deletion.

## 2 SN P systems with structural plasticity

Before introducing the notion of SN P systems with structural plasticity, we recall some prerequisites. It is useful for readers to have some familiarity with basic elements of formal language theory, e.g., from Rozenberg and Salomaa (1997), as well as with basic concepts and notions in SN P systems (Păun et al., 2010; Ionescu et al., 2006).

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

A regular expression over an alphabet $\Sigma$ is defined as follows:

1 $\lambda$ and each $a \in \Sigma$ is a regular expression

2 if $E_1, E_2$ are regular expressions over $\Sigma$, then $(E_1)(E_2)$ $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over $\Sigma$

3 nothing else is a regular expression over $\Sigma$.

With each regular expression $E$ we associate a language $L(E)$, defined in the following way:

1 $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in \Sigma$

2 $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions $E_1, E_2$ over $\Sigma$.

Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can also be written as $E^*$. By $\mathbb{N}$ we denote the set of natural numbers.

Let us now define SN P systems with structural plasticity. Such a system of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, out),$$

where

- $O = \{a\}$ is a singleton alphabet, where $a$ 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
  1 $n_i \in \mathbb{N}$ is the *initial number of spikes* contained in neuron $\sigma_i$;
  2 $R_i$ is a finite set of *rules* of following forms:
    a spiking rule: $E_1/a^{c_1} \rightarrow a; d$, where $E_1$ is a regular expression over $\{a\}$, $d \geq 0$ and $c_1 \geq 1$

b plasticity rule: $E_2/a^{c_2} \to \alpha(\beta, K, nei(i))$, where $E_2$ is a regular expression over $\{a\}$, $c_2 \geq 1$, $K \geq 0$, $\alpha \in \{+, -, \pm, \mp\}$, $\beta \in \{a, \lambda\}$ and $nei(i) \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_m\}$

c forgetting rule: $a^s \to \lambda$ for some $s \geq 1$ with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \to a; d$ or $E/a^c \to \alpha(\beta, K, nei(i))$ in $R_i$.

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ is the set of *synapses* between neurons;
- $out \in \{1, 2, \ldots, m\}$ indicates the *output* neuron.

Given neuron $\sigma_i$, we denote the set of neurons with neuron $\sigma_i$ as their presynaptic neuron by $pre(i)$, i.e., $pre(i) = \{\sigma_j \mid (i, j) \in syn\}$. Similarly, we denote the set of neurons which have neuron $\sigma_i$ as their postsynaptic neuron by $pos(i) = \{\sigma_j \mid (j, i) \in syn\}$.

For any spiking rule $E/a^c \to a; d$, if $L(E) = a^c$ holds, where $L(E)$ is the language associated with regular expressions $E$, then the spiking rule is called *pure*.

A spiking rule of the form $E_1/a^{c_1} \to a; d$ is applied as follows. If the neuron $\sigma_i$ contains $k_1$ spikes, and $a^{k_1} \in L(E_1), k_1 \geq c_1$, then the rule $E_1/a^{c_1} \to a; d \in R_i$ can be applied. This means that the neuron fires, consuming (removing) $c_1$ spikes (thus only $k_1 - c_1$ spikes remain in neuron $\sigma_i$) and producing one 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 synchronised). 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 in step $t$ and $d \geq 1$, then in steps $t$, $t + 1$, …, $t + d - 1$ the neuron is closed (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send several spikes along it, then these particular spikes are lost). In the step $t + d$, the neuron fires and becomes again open, so it can receive spikes (which can be used at step $t + d + 1$, when the neuron can again apply rules). If $E = a^c$, then the rule can be written in the simplified form $a^{c_1} \to a; d$; if $d = 0$, then the rule can be simply written as $E_1/a^{c_1} \to a$.

The plasticity rule of the form $E_2/a^{c_2} \to \alpha(\beta, K, nei(i))$ is applied in the following way. If the neuron $\sigma_i$ contains $k_2$ spikes, and $a^{k_2} \in L(E_2), k_2 \geq c_2$, then the rule $E_2/a^{c_2} \to \alpha(\beta, K, nei(i))$ can be applied. We refer to $nei(i) \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_m\}$ the set of 'neighbouring' neurons of $\sigma_i$ associated with the rule, defining the neurons to which neuron $\sigma_i$ can connect to or disconnect from using synapses. By using the rule, $c_2$ spikes are consumed from neuron $\sigma_i$ (remaining $k_2 - c_2$ spikes) and one of the following four operations is performed (depending on $\alpha$ and $\beta$).

1 $\alpha = +, \beta = a$: neuron $\sigma_i$ creates a new synapse to $K$ neurons from $nei(i)$; if $K < |nei(i) - pre(i)|$, then neuron $\sigma_i$ non-deterministically chooses $K$ neurons to create connections to, otherwise neuron $\sigma_i$ creates a synapse to all the neurons in $nei(i)$; $\beta = a$ indicates that when creating a synapse to the neurons from $nei(i)$, neuron $\sigma_i$ emits a spike to each of such neurons;

2 $\alpha = -, \beta = \lambda$: neuron $\sigma_i$ removes connections from $K$ neurons in $pre(i)$; if $K < |pre(i)|$, then neuron $\sigma_i$ non-deterministically chooses $K$ neurons to remove

connections from, otherwise neuron $\sigma_i$ delete connections to all the neurons in $pre(i)$; $\beta = \lambda$ indicates that when deleting synapses to the neurons from $pre(i)$, no spike consumption and production are involved in the process;

3 $\alpha = \pm, \beta = a$: neuron $\sigma_i$ creates a new synapse to $K$ neurons and then delete connections to $K$ neurons from $nei(i)$, this process is completed in a computation step; the respect operation of creating and deleting connections are as in operations (1) and (2); $\beta = a$ indicates that when creating a synapse to the neurons from $nei(i)$, neuron $\sigma_i$ emits a spike to each of such neurons;

4 $\alpha = \mp, \beta = a$: neuron $\sigma_i$ deletes connections to $K$ neurons and then creates connections to $K$ neurons from $nei(i)$, this process is completed in a computation step; the respect operation of creating and deleting connections are as in operations (1) and (2); $\beta = a$ indicates that when creating a synapse to the neurons from $nei(i)$, neuron $\sigma_i$ emits a spike to each of such neurons.

The rules of the form $a^s \to \lambda$ are *forgetting* rules. If neuron $\sigma_i$ contains exactly $s$ spikes, then the forgetting rule $a^s \to \lambda$ can be used, meaning that all $s$ spikes are removed from neuron $\sigma_i$. Note that, by definition, if a forgetting rule is applicable, then no spiking rule and plasticity rule can be applied, and vice versa.

In each time unit, if a neuron $\sigma_i$ can use one of its rules, then a rule from $R_i$ must be used. It is possible that there are more than one rule that can be used in a neuron at some moment, since spiking rules and plasticity rules may be associated with common regular languages (according to their regular expressions). In this case, the neuron will non-deterministically choose one of the enabled rules to use.

The system works sequentially on each neuron (at most one rule from each $R_i$ can be used), and in parallel at the level of the system, i.e., all the neurons having at least one enabled rule will use the rule in a parallel manner.

The *configuration* of the system is described by both the number of spikes associated with each neuron and by the number of steps to wait until it becomes open (this number is zero if the neuron is already open). Thus, the *initial configuration* is $\langle n_1/0, n_2/0, \ldots, n_m/0 \rangle$. Using the rules as described above, we 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 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.

With any spike train containing at least two spikes, the first two being emitted at steps $t_1$ and $t_2$, one associates a computation result in the form of the number $t_2 - t_1$; we say that this number is generated/computed by system $\Pi$. The set of all numbers computed in this way by $\Pi$ is denoted by $N_2(\Pi)$ (the subscript 2 indicates that the computation result is encoded by the time distance between the first two spikes of any computation).

We denote by $N_m SPSNP_2(\gamma, rule_r)$ the family of sets of numbers generated by SN P systems with structural plasticity of degree $m$, where $\gamma \subseteq \{+, -, \pm, \mp\}$ indicates the involved structural plasticity operations, $r$ is the maximal number of rules in any neuron, and the subscript 2 indicates that the system works in the generating mode and the computation result is encoded by the time distance between the first two spikes of any computation. If the parameter $m$ or $r$ is not bounded, then it is replaced with $*$.

In the following sections, SN P systems with structural plasticity are represented graphically. This is easier to understand than when given in in the formal definition. We use a rounded rectangle with the initial number of spikes and rules to represent a neuron and a directed edge connecting two neurons represents a synapse. The input/output neuron has an ingoing/outgoing arrow, suggesting their communication with the environment.

## 3 Universality result

In this section, we investigate the computational power of SN P systems with structural plasticity. We prove that SN P systems with structural plasticity are universal as number generating devices with the restriction that any neuron can only perform either synapse creation or synapse deletion, but not both of them, in a computation step (that is, the systems have the structural plasticity rules with only $\alpha = +$ or $-$, instead of $\pm$ and $\mp$). This result gives a positive answer to an open problem formulated in Francis et al. (2013).

The proof of the following theorem is based on the simulation of 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 (labelling an ADD instruction), $l_h$ is the halt label (assigned to instruction HALT), and $I$ is the set of instructions; each label from $H$ labels only one instruction from $I$, thus precisely identifying it. The instructions are of the following forms:

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

A register machine can work in the generating or accepting mode, and it is known that register machines compute all sets of numbers which are Turing computable, hence they characterise $NRE$ (the family of Turing computable sets of numbers) (see, e.g., Minsky, 1967).

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

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

*Theorem 1:* $N_2SPSNP(\gamma, rule_4) = NRE$ with $\gamma = \{+, -\}$.

*Proof:* It is enough to prove that $N_2SPSNP(\gamma, rule_2) \supseteq NRE$, since the converse inclusion is straightforward but cumbersome [for similar technical details, please refer to Chapter 7 in Păun (2002)].

For a given register machine $M = (m, H, l_0, l_h, I)$, we shall construct a specific SN P system with structural plasticity $\Pi$ to simulate $M$.

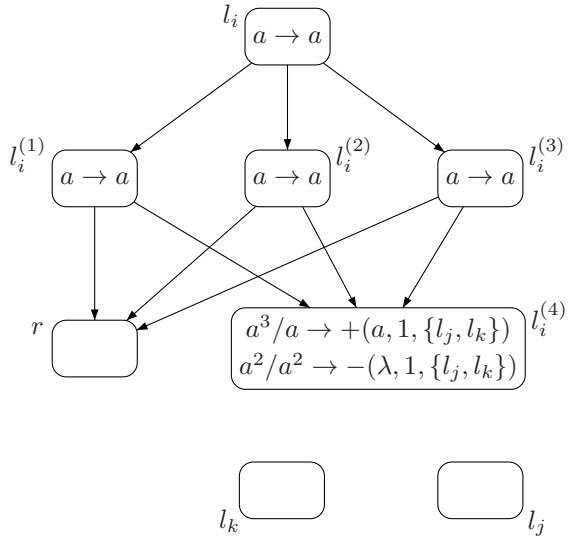
The system $\Pi$ consists of three types of modules – ADD modules, SUB modules, and FIN module shown in Figures 1, 2 and 3, respectively. The ADD and SUB modules are used to simulate the ADD and SUB instructions of $M$, respectively; FIN module is used to output a computation result.

In general, for each register $r$ of $M$, a neuron $\sigma_r$ is associated; the number stored in register $r$ is encoded by the number of spikes in neuron $\sigma_r$. Specifically, if register $r$ holds the number $n \geq 0$, then neuron $\sigma_r$ contains $3n$ spikes. For each label $l_i$ of an instruction in $M$, a neuron $\sigma_{l_i}$ is associated. During a computation, a neuron $\sigma_{l_i}$ having one spike inside will become active and start to simulate an instruction $l_i : (\texttt{OP}(r), l_j, l_k)$ of $M$: starting with neuron $\sigma_{l_i}$ activated, operating neuron $\sigma_r$ as requested by OP, then introducing one spike into neuron $\sigma_{l_j}$ or neuron $\sigma_{l_k}$, which becomes active in this way. When neuron $\sigma_{l_h}$ (associated with the label $l_h$ of the halting instruction of $M$) is activated, a computation in $M$ is completely simulated in $\Pi$; we will have that the output neuron $\sigma_{out}$ fires twice at step $t_1$ and $t_2$, respectively, and the interval $t_2 - t_1$ corresponds to the number stored in register 1 of $M$.

In what follows, the work of modules ADD, SUB, and FIN are described (that is, how they simulate the instructions of $M$ and output a computation result).

*Module ADD* (shown in Figure 1) – simulating an ADD instruction $l_i : (\texttt{ADD}\ (r), l_j, l_k)$.

**Figure 1** Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)




Let us assume that at step $t$, an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ has to be simulated, with one spike present in neuron $\sigma_{l_i}$ and no spike in any other neurons, except for neurons $\sigma_r$ associated with registers $r$ (the spikes in neuron $\sigma_r$ encodes the number stored in the register $r$). Having one spike inside, neuron $\sigma_{l_i}$ fires at step $t$ sending one spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$, respectively. Neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ become active and fire at step $t + 1$, sending three spikes to each of neurons $\sigma_r$ and $\sigma_{l_i^{(4)}}$. In this way,

the number of spikes in neuron $\sigma_r$ increases by three simulating that the number stored in register $r$ is increased by one. In neuron $\sigma_{l_i^{(4)}}$, plasticity rule $a^3/a \to +(a, 1, \{l_j, l_k\})$ can be used. By this rule, a new synapse will be non-deterministically created to one of the two neurons $\sigma_{l_j}$ and $\sigma_{l_k}$,

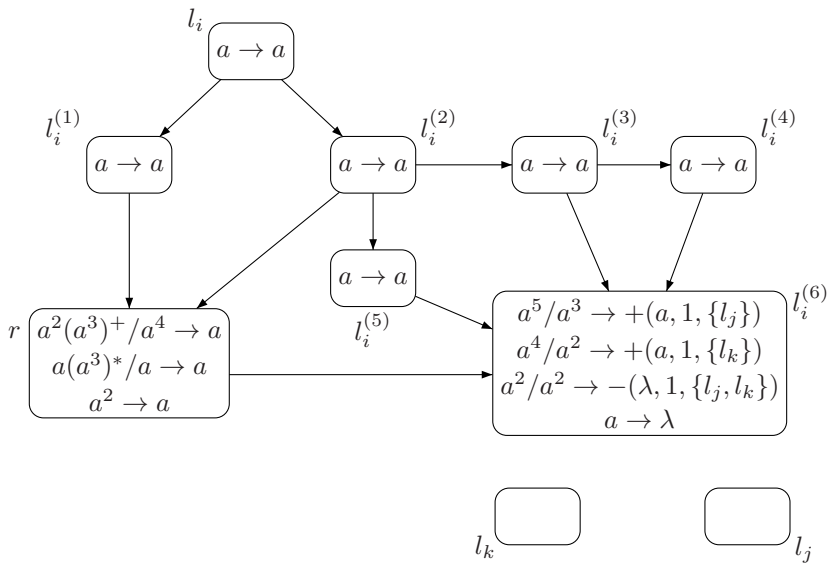
1 If neuron $\sigma_{l_i^{(4)}}$ chooses to create connection to neuron $\sigma_{l_j}$ at step $t+2$, then it produces a synapse to neuron $\sigma_{l_j}$ sending a spike to neuron $\sigma_{l_j}$. In this case, the number of spikes in neuron $\sigma_{l_i^{(4)}}$ becomes 2 such that plasticity rule $a^2/a^2 \to -(\lambda, 1, \{l_j, l_k\})$ is enabled and applied at step $t+3$ removing synapse $(l_i^{(4)}, l_j)$. The module returns to the initial topological structure. At step $t+3$, neuron $\sigma_{l_j}$ has a spike inside and fires, which means that the system $\Pi$ starts to simulate the instruction with label $l_j$.

2 If neuron $\sigma_{l_i^{(4)}}$ chooses to create connection to neuron $\sigma_{l_k}$, then it creates a synapse to neuron $\sigma_{l_k}$ and sends a spike to neuron $\sigma_{l_k}$ at step $t+2$. At step $t+3$, plasticity rule $a^2/a^2 \to -(\lambda, 1, \{l_j, l_k\})$ in neuron $\sigma_{l_i^{(4)}}$ is applied to remove synapse $(l_i^{(4)}, l_k)$. The module returns to the initial topological structure, and neuron $\sigma_{l_k}$ fires at step $t+3$, which means that the system $\Pi$ starts to simulate the instruction with label $l_k$.

Therefore, from firing neuron $\sigma_{l_i}$, three spikes are added to neuron $\sigma_r$, and one of neurons $\sigma_{l_j}$ and $\sigma_{l_k}$ non-deterministically fires, which correctly simulates the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

*Module SUB* (shown in Figure 2) – simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$.

**Figure 2** Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)



Recalling the fact that the first register of register machine $M$ is never decremented during a computation (i.e., its content can only increase). So, in the SUB modules constructed here, for each neuron $\sigma_r$ corresponding to register $r$ of $M$, we have $r \neq 1$.

Assume that, at step $t$, neuron $\sigma_{l_i}$ contains a spike, then it is activated and fires, and both neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ receive one spike. Neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at step $t+1$, sending two spikes to neuron $\sigma_r$ and one spike to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(5)}}$. At step $t+2$, neuron $\sigma_r$ becomes active by spiking rule $a^2(a^3)^+/a^4 \to a$ or $a^2 \to a$, which depends on the number of spikes in neuron $\sigma_r$. There are the following two cases.

1 At step $t+2$, neuron $\sigma_r$ has $3n+2$ ($n>0$) spikes (corresponding to the fact that the number stored in register $r$ is $n$, and $n>0$). In this case, in neuron $\sigma_r$ spiking rule $a^2(a^3)^+/a^4 \to a$ fires consuming four spikes (the number of spikes in neuron $\sigma_r$ becomes $3n+2-4=3(n-1)+1$), and sending a spike to neuron $\sigma_{l_i^{(6)}}$. Meanwhile, neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(5)}}$ fire, sending two spikes to neuron $\sigma_{l_i^{(6)}}$. At step $t+3$, neuron $\sigma_r$ fires again by rule $a(a^3)^*/a \to a$, sending one more spike to neuron $\sigma_{l_i^{(6)}}$; and neuron $\sigma_{l_i^{(3)}}$ fires sending a spike to neuron $\sigma_{l_i^{(6)}}$. In this way, the number of spikes in neuron $\sigma_r$ becomes $3(n-1)$ simulating that the number in register $r$ is decreased by one; and neuron $\sigma_{l_i^{(6)}}$ accumulates five spikes inside. At step $t+4$, plasticity rule $a^5/a^3 \to +(a,1,\{l_j\})$ in neuron $\sigma_{l_i^{(6)}}$ is applied consuming three spikes, creating a synapse from neuron $\sigma_{l_i^{(6)}}$ to neuron $\sigma_{l_j}$ and sending a spike to neuron $\sigma_{l_j}$. In the next step, plasticity rule $a^2/a^2 \to -(\lambda,1,\{l_j,l_k\})$ in neuron $\sigma_{l_i^{(6)}}$ is enabled and applied, deleting synapse $(l_i^{(6)},l_j)$ to return the SUB module to initial topological structure. With one spike inside, neuron $\sigma_{l_j}$ is enabled and fires at step $t+5$, which means that the system $\Pi$ starts to simulate instruction $l_j$ of $M$.

2 At step $t+2$, neuron $\sigma_r$ has two spikes (corresponding to the fact that the number stored in register $r$ is 0). In this case, neuron $\sigma_r$ fires by spiking rule $a^2 \to a$ sending a spike to neuron $\sigma_{l_i^{(6)}}$. At the same moment, each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(5)}}$ send a spike to neuron $\sigma_{l_i^{(6)}}$. At step $t+3$, neuron $\sigma_{l_i^{(4)}}$ fires sending a spike to neuron $\sigma_{l_i^{(6)}}$. In this way, neuron $\sigma_{l_i^{(6)}}$ accumulates four spikes inside, and plasticity rule $a^4/a^2 \to +(a,1,\{l_k\})$ is enabled and applied at step $t+4$, by which neuron $\sigma_{l_i^{(6)}}$ creates a new synapse connected to neuron $\sigma_{l_k}$ and sends a spike to neuron $\sigma_{l_k}$. One step later, plasticity rule $a^2/a^2 \to -(\lambda,1,\{l_j,l_k\})$ in neuron $\sigma_{l_i^{(6)}}$ is enabled and applied, deleting synapse $(l_i^{(6)},l_k)$ to return the SUB module to initial topological structure. With one spike inside, neuron $\sigma_{l_k}$ fires at step $t+5$. When neuron $\sigma_{l_k}$ fires, the system $\Pi$ starts to simulate instruction $l_k$ of $M$.

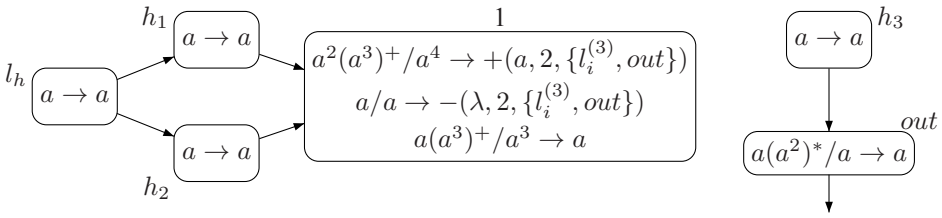
Note that there is no interference between the ADD modules and the SUB modules, other than correctly firing the neurons $\sigma_{l_j}$ or $\sigma_{l_k}$. However, it is possible to have interference between two SUB modules. Specifically, if there are several SUB instructions $l_u$ that act on register $r$, then neuron $\sigma_r$ has synapse connection to all neurons $\sigma_{l_u^{(6)}}$. When a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with $l_u$ ($l_u \neq l_i$) all neurons receive no spike except that neurons

$\sigma_{l_u^{(6)}}$ may receive one spike (corresponding the fact the number in register $r$ is 0) or two spikes in separated two steps (corresponding the fact the number in register $r$ is $n > 0$). The spike will be removed by forgetting rule $a \to \lambda$ in at most two steps. So, the interference among SUB modules will not cause undesired steps in $\Pi$ (i.e., steps that do not correspond to correct simulations of instructions of $M$).

Therefore, the simulation of SUB instruction is correct: system $\Pi$ starts from neuron $\sigma_{l_i}$ and ends in neuron $\sigma_{l_j}$ (if the number stored in register $r$ is greater than 0 and decreased by one), or in neuron $\sigma_{l_k}$ (if the number stored in register $r$ is 0).

*Module FIN* (shown in Figure 3) – outputting the result of computation.

**Figure 3** Module FIN (outputting the computation result)




Assume now that the computation in $M$ halts, which means that the halting instruction is reached. This means that neuron $\sigma_{l_h}$ in $\Pi$ gets one spike and fires by the rule $a \to a$. At that moment, neuron $\sigma_1$ contains $3n$ spikes, for the number $n$ stored in register 1 of $M$. When neuron $\sigma_{l_h}$ fires, each of neurons $\sigma_{h_1}$ and $\sigma_{h_2}$ receives one spike. With one spike inside, neurons $\sigma_{h_1}$ and $\sigma_{h_2}$ fire sending two spikes to neuron $\sigma_1$. The number of spikes in neuron $\sigma_1$ becomes $3n + 2$ such that plasticity rule $a^2(a^3)^+/a^4 \to +(a, 2, \{l_i^{(3)}, out\})$ is enabled and applied at step $t + 2$, creating two new synapses $(1, h_3)$ and $(1, out)$ and sending a spike to neurons $\sigma_{h_3}$ and $\sigma_{out}$, respectively. Having a spike inside, neuron $\sigma_{out}$ fires for the first time at step $t + 3$ sending a spike into the environment. The number of spikes in neuron $\sigma_1$ becomes $3(n - 1) + 1$, thus spiking rule $a(a^3)^+/a^3 \to a$ is enabled and applied in each step from step $t + 3$ to step $t + n + 1$. During the time period, neuron $\sigma_{out}$ receives two spikes in each step and keeps inactive. At step $t + n + 2$, neuron $\sigma_1$ contains one spike inside and spiking rule $a(a^3)^+/a^3 \to a$ cannot be applied, but plasticity rule $a/a \to -(\lambda, 2, \{l_i^{(3)}, out\})$ is enabled and applied to delete synapses $(1, h_3)$ and $(1, out)$. In this way, the FIN module returns to the initial topological structure.

At step $t + n + 2$, neuron $\sigma_{h_3}$ fires sending a spike to neuron $\sigma_{out}$. After receiving this spike, neuron $\sigma_{out}$ fires for the second time at step $t + n + 3$, sending the second spike into the environment. The interval between these two spikes sent to the environment is $(t + n + 3) - (t + 3) = n$, which is exactly the number stored in register 1 of register $M$ 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 SN P system with structural plasticity $\Pi$, i.e., $N(M) \subseteq N(\Pi)$.

We can check that in the system $\Pi$, each neuron contains at most four rules and the operation associated with structural plasticity rules has $\alpha = +$ or $-$, that is, the involved

structural plasticity operations $\gamma = \{+, -\}$. Therefore, $N_2SPSNP(\gamma, rule_4) = NRE$ with $\gamma = \{+, -\}$.

## 4 Conclusions and discussion

In this work, we investigated the computational power of SN P systems with structural plasticity. A normal form of universal SN P systems with structural plasticity was given. Specifically, we proved that SN P systems with structural plasticity can compute any set of Turing computable natural numbers with the restriction that any neuron can either create or delete synapses, but not both of creating and deleting synapses in a computation step. This result gives a positive answer to an open problem formulated in Francis et al. (2013). These results may give some hints to construct universal SN P systems having flexible and simple topological structure by means of synaptogenesis and synapse deletion.

It deserves for further research whether there are improved normal forms of universal SN P systems with structural plasticity. For example, pure form of rules was proved to be sufficient for achieving universality of SN P system with anti-spikes (Song et al., 2012); it remains open whether there exist Turing universal SN P systems with structural plasticity and pure form of spiking rules.

In the SN P systems constructed in the proof of Theorem 3, most of the synapses are initially constructed and plasticity rules are only used in some particular neurons to generate and delete synapses. It is of interest to construct Turing universal SN P systems without initial synapses; that is, all synapses involved in the computation are created when they are required, and synapses can be removed during a computation such that the system always has a few synapses.

## References

Cavaliere, M., Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M. and Woodworth, S. (2009) 'Asynchronous spiking neural P systems', *Theoretical Computer Science*, Vol. 410, No. 24, pp.2352–2364.

Chen, H., Freund, R., Ionescu, M., Păun, G. and Pérez-Jiménez, M.J. (2007) 'On string languages generated by spiking neural P systems', *Fundamenta Informaticae*, Vol. 75, No. 1, pp.141–162.

Francis, G.C.C., Henry, N.A. and Gyen, N.I. (2013) 'Spiking neural P systems with structural plasticity', in Zhang Gexiang, Wang Jun, Cheng Jixiang, and Wang Tao (Eds.): *Pre-proceedings of 2nd Asian Conference on Membrane Computing*, pp.13–26, Chengdu, China, October.

Ibarra, O.H., Păun, A., Păun, G., Rodríguez-Patón, A., Sosík, P. and Woodworth, S. (2007) 'Normal forms for spiking neural P systems', *Theoretical Computer Science*, Vol. 372, No. 2, pp.196–217.

Ionescu, M. and Sburlan, D. (2012) 'Some applications of spiking neural p systems', *Computing and Informatics*, Vol. 27, No. 3, pp.515–528.

Ionescu, M., Păun, G. and Yokomori, T. (2006) 'Spiking neural P systems', *Fundamenta Informaticae*, Vol. 71, No. 2, pp.279–308.

Leporati, A., Mauri, G., Zandron, C., Păun, G. and Pérez-Jiménez, M.J. (2009) 'Uniform solutions to SAT and Subset Sum by spiking neural P systems', *Natural Computing*, Vol. 8, No. 4, pp.681–702.

Macías-Ramos, L.F. and Pérez-Jiménez, M.J. (2013) 'Spiking neural P systems with functional astrocytes', in *Lecture Notes in Computer Science*, Vol. 7762, pp.228–242, Springer.

Martín-Vide, C., Păun, G., Pazos, J. and Rodrıguez-Patón, A. (2003) 'Tissue P systems', *Theoretical Computer Science*, Vol. 296, No. 2, pp.295–326.

Metta, V.P. and Kelemenová, A. (2014) 'Universality of spiking neural P systems with anti-spikes', in *Theory and Applications of Models of Computation*, pp.352–365, Springer.

Minsky, M.L. (1967) *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, NJ.

Păun, G., Rozenberg, G. and Salomaa, A. (2010) *The Oxford Handbook of Membrane Computing*, Oxford University Press, Inc., Oxford.

Păun, G. (2002) *Membrane Computing: An Introduction*, Springer, Berlin.

Păun, A. and Păun, G. (2007) 'Small universal spiking neural P systems', *BioSystems*, Vol. 90, No. 1, pp.48–60.

Păun, G. (2000) 'Computing with membranes', *Journal of Computer and System Sciences*, Vol. 61, No. 1, pp.108–143.

Păun, G. (2007) 'Spiking neural P systems with astrocyte-like control', *Journal of Universal Computer Science*, Vol. 13, No. 11, pp.1707–1721.

Pan, L. and Păun, G. (2009) 'Spiking neural P systems with anti-spikes', *International Journal of Computers, Communications & Control*, Vol. 4, No. 3, pp.273–283.

Pan, L. and Păun, G. (2010) 'Spiking neural P systems: an improved normal form', *Theoretical Computer Science*, Vol. 411, No. 6, pp.906–918.

Pan, L. and Zeng, X. (2010) 'A note on small universal spiking neural P systems', in *Lecture Notes in Computer Science*, Vol. 5957, pp.436–447, Springer.

Pan, L., Păun, G. and Pérez-Jiménez, M.J. (2011) 'Spiking neural P systems with neuron division and budding', *Science China Information Sciences*, Vol. 54, No. 8, pp.1596–1607.

Pan, L., Wang, J. and Hoogeboom, H.J. (2012) 'Spiking neural P systems with astrocytes', *Neural Computation*, Vol. 24, No. 3, pp.805–825.

Rozenberg, G. and Salomaa, A. (1997) *Handbook of Formal Languages*, Vol. 3. Springer-Verlag, Berlin.

Song, T. and Wang, X. (2014) 'Homogenous spiking neural P systems with inhibitory synapses', *Neural Processing Letters*, DOI: 10.1007/s11063-014-9352-y.

Song, T., Pan, L., Wang, J., Venkat, I., Subramanian, K.G. and Abdullah, R. (2012) 'Normal forms of spiking neural P systems with anti-spikes', *IEEE Transactions on NanoBioscience*, Vol. 11, No. 4, pp.352–359.

Song, T., Pan, L. and Păun, G. (2013a) 'Asynchronous spiking neural P systems with local synchronization', *Information Sciences*, Vol. 219, pp.197–207.

Song, T., Shi, X. and Xu, J. (2013b) 'Reversible spiking neural P systems', *Frontiers of Computer Science*, Vol. 7, No. 3, pp.350–358.

Song, T., Pan, L. and Păun, G. (2014a) 'Spiking neural P systems with rules on synapses', *Theoretical Computer Science*, Vol. 529, pp.82–95.

Song, T., Wang, X., Zhang, Z. and Chen, Z. (2014b) 'Homogenous spiking neural P systems with anti-spikes', *Neural Computing and Applications*, Vol. 24, Nos. 7–8, pp.1833–1841.

Zhang, X., Zeng, X. and Pan, L. (2008a) 'On string languages generated by spiking neural P systems with exhaustive use of rules', *Natural Computing*, Vol. 7, No. 4, pp.535–549.

Zhang, X., Zeng, X. and Pan, L. (2008b) 'Smaller universal spiking neural P systems', *Fundamenta Informaticae*, Vol. 87, No. 1, pp.117–136.

Zhang, X., Zeng, X., Luo, B. and Xu, J. (2012) 'Several applications of spiking neural P systems with weights', *Journal of Computational and Theoretical Nanoscience*, Vol. 9, No. 6, pp.769–777.