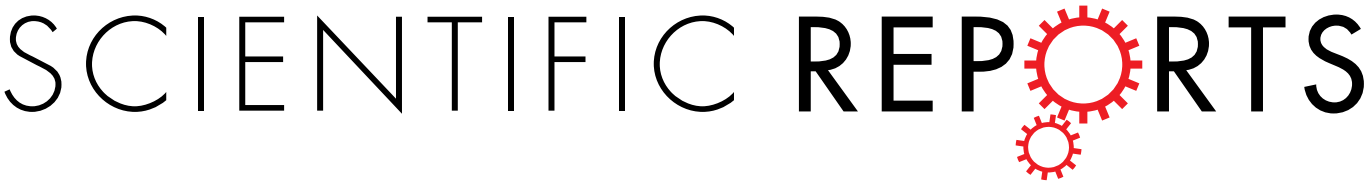

OPEN

On the Computational Power of Spiking Neural P Systems with Self-Organization

Received: 01 March 2016
Accepted: 23 May 2016
Published: 10 June 2016

Xun Wang[1], Tao Song[1,2], Faming Gong[1] & Pan Zheng[2]

Neural-like computing models are versatile computing mechanisms in the field of artificial intelligence. Spiking neural P systems (SN P systems for short) are one of the recently developed spiking neural network models inspired by the way neurons communicate. The communications among neurons are essentially achieved by spikes, i. e. short electrical pulses. In terms of motivation, SN P systems fall into the third generation of neural network models. In this study, a novel variant of SN P systems, namely SN P systems with self-organization, is introduced, and the computational power of the system is investigated and evaluated. It is proved that SN P systems with self-organization are capable of computing and accept the family of sets of Turing computable natural numbers. Moreover, with 87 neurons the system can compute any Turing computable recursive function, thus achieves Turing universality. These results demonstrate promising initiatives to solve an open problem arisen by Gh Păun.

In the central nervous system, there are abundant amount of computational intelligence precipitated throughout millions of years of evolution. The computational intelligence has provided plenty of inspirations to construct powerful computing models and algorithms[1–3]. Neural-like computing models are a class of powerful models inspired by the way how neurons communicate. The communication among neurons is essentially achieved by spikes, i.e. short electrical pulses. The biological phenomenon has been intensively investigated in the field of neural computation[4]. Using different mathematic approaches to describe neural spiking behaviours, various neural-like computing models have been proposed, such as artificial neural networks[5] and spiking neural networks[6]. In the field of membrane computing, a kind of distributed and parallel neural-like computation model, named spiking neural P systems (SN P systems), were proposed in 2006[7]. SN P systems are widely considered as a promising variant of the third generation of neural network models[8].

Generally, an SN P system can be represented by a directed graph, where neurons are placed in nodes and the synapses are denoted using arcs. Every neuron can contain a number of spikes and a set of firing (or spiking) rules. Following the firing rules, a neuron can send information encoded in spikes to other neurons. Input neurons read spikes from the environment, and output neurons emit spikes into the environment. The computation result can be embodied in various ways. One of the common approaches is the time elapsed between the first two consecutive spikes sent into the environment[9,10] and the total number of spikes emitted into the environment[11–13].

For the past decade, there have been quite a few research efforts put forward to SN P systems. Notably, SN P systems can generate and accept the sets of Turing computable natural numbers[14], generate the recursively enumerable languages[15,16] and compute the sets of Turing computable functions[17]. Inspired by different biological phenomena and mathematical motivations, lots of variants of SN P systems have been proposed, such as SN P systems with anti-spikes[18,19], SN P systems with weight[20], SN P systems with astrocyte[21], homogenous SN P systems[22,23], SN P systems with threshold[24], fuzzy SN P systems[25,26], sequential SN P systems[27], SN P systems with rules on synapses[28], SN P systems with structural plasticity[29]. For applications, SN P systems are used to design logic gates, logic circuites[30] and operating systems[31], perform basic arithmetic operations[32,33], solve combinatorial optimization problems[34], diagnose fault of electric power systems[35].

SN P systems are known as a class of neural-like computing models under the framework of membrane computing[36]. Spiking neural network (shortly named SNN) is a well known candidate of siking neural network

[1]College of Computer and Communication Engineering, China University of Petroleum, Qingdao 266580, Shandong, China. [2]Faculty of Engineering, Computing and Science, Swinburne University of Technology Sarawak Campus, Kuching, 93350, Malaysia. Correspondence and requests for materials should be addressed to T.S. (email: tsong@upc.edu.cn or tsong@swinburne.edu.my)

models[37], which incorporates the concept of time into their operating model, besides neuronal and synaptic state in general artificial neural networks, The neuron in SNN cannot fire at each propagation cycle, but only when a membrane potential reaches a specific value. When a neuron fires, it generates a signal which travels to other neurons which, in turn, increase or decrease their potentials in accordance with this signal. In SN P systems, spiking rules, denoted by formal production in grammar theory of formal languages, is used to describe the neuron's spiking behaviour, which determine the conditions of triggering spiking, the number of spikes consumed, and the number of spikes emitting to the neighboring neurons. The spikes from different neurons can be accumulated in the target neuron for further spiking. In terms of motivation of models, SN P systems also fall into the spiking neural network models, i.e., the third generation of neural network models.

Since SN P systems have more fundamental data structure (spike trains, i.e., binary strings), it performs well in achieving significant computation power with using a small number of units (neurons). It was proved by Gh Păun that 49 neurons are sufficient for SN P systems to achieve Turing universality. But, for conventional artificial neural networks, it was shown that 886 sigmoid function based processors are needed to achieve Turing universality[38].

In the nervous system, synaptic plasticity forms the cell assemblies with the self-organization of neurons, which induces ordered or even synchronized neural dynamics replicating basic processes of long-term memory[39,40]. The self-organizing principle in the developing nervous system and its importance for preserving and continuing neural system development provide us insights on how neural-like networks might be reorganized and configured in response to environment changes. Enlightened by the biological fact, self-organizing artificial neural networks with unsupervised and supervised learning have been proposed and gain their popularity for visualisation and classification[41,42]. It is still an open problem as formulated by Gh Păun in ref. 43, to construct SN P systems with self-organization and to use the system to perform possible computer vision and pattern recognition tasks.

Results

In this research, a novel variant of SN P systems, namely SN P systems with self-organization, is proposed and developed. The system initially has no synapse, but the synapses can be dynamically formed during the computation, which exhibits the self-organization behaviour. In the system, creation and deletion rules are used to create and delete synapses. The applications of synapse creation and deletion rules are controlled by the states of the involved neurons, i.e., the number of spikes contained in the neurons. The computational power of the system is investigated as well. As a result, it demonstrates that SN P systems with self-organization can compute and accept any set of Turing computable natural numbers. Moreover, with 87 neurons, the system can compute any Turing computable recursive function, ergo achieves Turing universality.

Before stating the results in mathematical forms, some notations should be introduced. $N_mSPSO_{all}(cre_h, del_g, rule_r)$ (resp. $N_mSPSO_{acc}(cre_{h'}, del_{g'}, rule_{r'})$) denotes the family of sets of numbers computed (resp. accepted) by SN P systems with self-organization of degree m, where h (resp. h') indicates the maximal number of synapses that can be created using a synapse creation rule, g (resp. g') is the maximal number of synapses that can be deleted by using a synapse deletion rule, r (resp. r') is the maximal number of rules in each neuron, and the subscript *all* indicates the computation result is encoded by the number of spikes emitted into the environment (resp. the subscript *acc* indicates the system works in the accepting mode). If the parameters are not bounded, i.e., there is no limit imposed on them, then they are replaced with *. *NRE* denotes the family of Turing computable sets of numbers[44].

The main results of this work can be mathematically depicted by the following theorems.

Theorem 1. $N_*SPSO_{all}(cre_*, del_*, rule_5) = NRE$.
Theorem 2. $N_*SPSO_{acc}(cre_*, del_*, rule_5) = NRE$.
Theorem 3. *There is a universal SN P system with self-organization having 87 neurons for computing functions.*

These results show that SN P systems with self-organization are powerful computing models, i.e., they are capable of doing what Turing machine can do. Also, they provide potential and theoretical feasibility of using SN P systems to solve real-life problems, such as pattern recognition and classification.

In SN P system with self-organization, it has no initially designed synapses. The synapses can be created or deleted according to the information contained in involved neurons during the computation. In previous work, it was found that the information diversing ability of synapses had some programable feature for SN P systems, but the computation power of SN P systems without initial synapses is an open problem. Although this is not the first time the feature of creating or deleting synapses investigated in SN P systems, see e.g. SN P systems with structural plasticity, it is quite the first attempt to construct SN P systems has no initial synapses.

Methods

In this section, it starts by the mathematical definition of SN P system with self-organization, and then the computation power of SN P systems with self-organization is investigated as number generator, acceptor and function computing devices. It is proved in constructive ways that SN P systems with self-organization can compute and accept the family of sets of Turing computable natural numbers. With 87 neurons, such system can compute any Turing computable recursive function.

Spiking Neural P Systems with Self-Organization. Before introducing the definition of SN P system with self-organization, some prerequisites of basic concepts of formal language theory[45] are recalled.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and 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 $\{a\}^*$, $\{a\}^+$. A regular expression over an alphabet V is defined as follows: (1) λ and each $a \in V$ is

a regular expression; (2) if E_1 and 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; (3) nothing else is a regular expression over V.

For each regular expression E, a language $L(E)$ is associated, defined in the following way: (1) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (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 V. Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can also be written as E^*. By *NRE* we denote the family of Turing computable sets of numbers. (*NRE* is the family of length sets of recursively enumerable languages–those recognized by Turing machines).

An SN P system with self-organization of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, s_1, s_2, \ldots, s_m, syn_0, in, out)$$

where

- $O = \{a\}$ is a singleton, where a is called the 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
  - $n_i \in \mathbb{N}$ is the initial number of spikes contained in neuron σ_i;
  - R_i is a finite set of rules in neuron σ_i of the following three forms:
    (1) spiking rule: $E/a^c \rightarrow a^p; d$, where E is a regular expression over O, $d \geq 0$ and $c \geq p \geq 0$;
    (2) synapse creation rule: $E'/a^{c'} \rightarrow +(a^{p'}, cre(i))$, where E' is a regular expression over O, $cre(i) \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_m\}/\{\sigma_i\}$ and $c' \geq p' > 1$;
    (3) synapse deletion rule: $E''/a^{c''} \rightarrow -(\lambda, del(i))$, where E'' is a regular expression over O, $del(i) \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_m\}/\{\sigma_i\}$ and $c'' \geq 1$;
- $syn_0 = \emptyset$ is the initial set of synapses, which means no synapse is initially set; at any moment t, the set of synapses is denoted by $syn_t \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$.
- $in, out \in \{1, 2, \ldots, m\}$ indicates the input and output neuron, respectively.

A spiking rule of the form $E/a^c \rightarrow a^p; d$ is applied as follows. If neuron σ_i contains k spikes, and $a^k \in L(E)$, $k \geq c$, then rule $E/a^c \rightarrow a^p; d \in R_i$ can be applied. It means that c spikes are consumed and removed from neuron σ_i, i.e., $k - c$ spikes are remained, while the neuron emits p spikes to its neighboring neurons after d steps. (It is a common practice in membrane computing to have a global clock defined. The clock is used to mark the time of the whole system and ensure the system synchronization.) If $d = 0$, then the p spikes are emitted out immediately, if $d = 1$, then the p spikes are emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps t, $t + 1$, ..., $t + d - 1$ the neuron is closed (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron tries to send spikes to a neuron in close status, then these particular spikes will be lost). In the step $t + d$, the neuron fires and regains open status, so it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). It is possible that p is associated with value 0. In this case, neuron σ_i consumes c spikes without emitting any spike. Spiking rule with $p = 0$ is also called forgetting rule, by which a pre-defined number of spikes can be removed out of the neuron. If $E = a^c$, then the rule can be written in the simplified form $a^c \rightarrow a^p; d$, and if $d = 0$, then the rule can be simply written as $E/a^c \rightarrow a^p$.

Synapse creation and deletion rules are used to create and delete synapses during the computation. Synapse creation rule $E'/a^{c'} \rightarrow +(a^{p'}, cre(i))$ is applied as follows. If neuron σ_i has k' spikes such that $a^{k'} \in L(E')$, $k' \geq c'$, then the synapse creation rule is applied with consuming c' spikes, creating synapses to connect neuron σ_i to each neuron in $cre(i)$ and emitting p' spikes to each neuron in $cre(i)$. If neuron σ_i has k'' spikes such that $a^{k''} \in L(E'')$ and $k'' \geq c''$, then synapse deletion rule $E''/a^{c''} \rightarrow -(\lambda, del(i))$ is applied, removing c'' spikes from neuron σ_i and deleting all the synapses connecting neuron σ_i to the neurons from $del(i)$. With the synapse creation and deletion rules, E' and E'' are regular expressions over $O = \{a\}$, which regulate the application of synapse creation and deletion rules. This means that synapse creation and deletion rules can be used if and only if the neuron contain some particular numbers of spikes, i.e., the neuron is in some specific states. With the applications of synapse creation and deletion rules the system can dynamically rebuild its topological structure during the computation, which is herein defined as self-organization.

One neuron is specified as the input neuron, through which the system can read spikes from the environment. The output neuron has a synapse creation rule of the form $E'/a^{c'} \rightarrow +(a^{p'}, \{0\})$, where the environment is labelled by 0. By using the rule, the output neuron creates a synapse pointing to the environment, and then it can emit spikes into the environment along the created synapse.

For each time step, as long as there is one available rule in R_i, neuron σ_i must apply the rule. It is possible that there are more than one rule that can be used in a neuron at some moment, since spiking rules, synapse creation rules and synapse deletion rules may be associated with regular languages (according to their regular expressions). In this case, the neuron will non-deterministically uses one of the enabled rules. The system works sequentially in each neuron (at most one rule from each R_i can be used), and if parallelism is designed for the system, all the neurons at the same system level have at least one enabled rule activated.

The configuration of the system at certain moment is defined by three major factors which are the number of spikes contained in each neuron, the number of steps to wait until it becomes open and the current set of synapses. With the notion, the initial configuration of the system is $\langle n_1/0, n_2/0, \ldots, n_m/0, \emptyset \rangle$. Using the spiking, forgetting, synapse creation and deletion 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, also called successful, if it reaches a configuration where no rule can be applied in any neuron in the system. For each

successful computation of the system, a computation result is generated, which is total the number of spikes sent to the environment by the output neuron.

System Π generates a number n as follows. The computation of the system starts from the initial configuration and finally halts, emitting totally n spikes to the environment. The set of all numbers computed in this way by Π is denoted by $N_{all}(\Pi)$ (the subscript *all* indicates that the computation result is the total number of spikes emitted into the environment by the system). System Π can also work in the accepting mode. A number n is read through input neurons from the environment in form of spike train $10^{n-1}1$, which will be stored in a specified neuron σ_1 in the form of $f(n)$ spikes. If the computation eventually halts, then number n is said to be accepted by Π. The set of numbers accepted by Π is denoted by $N_{acc}(\Pi)$.

It is denoted by $N_mSPSO_{all}(cre_h, del_g, rule_r)$ (resp. $N_mSPSO_{acc}(cre_{h'}, del_{g'}, rule_{r'})$) the family of sets of numbers computed (resp. accepted) by SN P systems with self-organization of degree m, where h (resp. h') indicates the maximal number of synapses that can be created with using a synapse creation rule, g (resp. g') is the maximal number of synapses that can be deleted with using a synapse deletion rule, r (resp. r') is the maximal number of rules in each neuron, and the subscript *all* indicates the computation result is encoded by the number of spikes emitted into the environment (resp. the subscript *acc* indicates the system works in a accepting mode). If the parameters are not bounded, i.e., there is no limit imposed on them, then they are replaced with *.

In order to compute a function $f: \mathbf{N}^k \to \mathbf{N}$ by SN P systems with self-organization, k natural numbers $n_1, n_2, \ldots, n_k$ are introduced in the system by reading from the environment a spike train (which is a binary sequence) $z = 10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$. The input neuron has a synapse pointing from the environment, by which the spikes can enter it. The input neuron reads a spike in each step corresponding to a digit 1 from the string z; otherwise, no spike is received. Note that exactly $k+1$ spikes are introduced into the system through the input neuron, i.e., after the last spike, it is assumed that no further spike is coming to the input neuron. The output neuron has a synapse pointing to the environment from it, by which the spikes can be emitted to the environment. The result of the computation is the total number of spikes emitted into the environment by the output neuron, hence producing r spikes with $r = f(n_1, n_2, \ldots, n_k)$.

SN P systems with self-organization can be represented graphically, which is easier to understand than that in a symbolic way. A rounded rectangle with the initial number of spikes and rules is used to represent a neuron and a directed edge connecting two neurons represents a synapse.

In the following proofs, the notion of register machine is used. 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, 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 precisele following forms:

- l_i: (ADD(r), l_j, l_k) (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- l_i: (SUB(r), l_j, l_k) (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise, go to the instruction with label l_k),
- l_h: HALT (the halt instruction).

As number generator. A register machine M computes a number n as follows. It starts by using initial instruction l_0 with all registers storing number 0. When it reaches halt instruction l_h, the number stored in register 1 is called the number generated or computed by register machine M. The set of numbers generated or computed by register machine 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*, i.e., $N(M) = NRE$, where *NRE* is the family of Turing computable sets of numbers[44].

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). When the power of two number generating devices D_1 and D_2 are compared, 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 in language and automata theory).

Theorem 4. $N_*SPSO_{all}(cre_*, del_*, rule_5) = NRE$.

Proof. It only has to prove $NRE \subseteq N_*SPSO_{all}(cre_*, del_*, rule_5)$, since the converse inclusion is straightforward from the Turing-Church thesis (or it can be proved by the similar technical details in Section 8.1 in ref. 46, but is cumbersome). To achieve this, we use the characterization of *NRE* by means of register machines in the generative mode. Let us consider a register machine $M = (m, H, l_0, l_h, I)$ defined above. It is assumed that register 1 of M is the output register, which is never decremented during the computation. For each register r of M, let s_r be the number of instructions of the form l_i: (SUB(r), l_j, l_k), i.e., the number of SUB instructions acting on register r. If there is no such SUB instruction, then $s_r = 0$, which is the case for the first register $r = 1$. In what follows, a specific SN P system with self-organization Π is constructed to simulate register machine M.

System Π consists of three modules–ADD, SUB and FIN modules. The ADD and SUB modules are used to simulate the operations of ADD and SUB instructions of M; and the FIN module is used to output a computation result.

In general, with any register r of M, a neuron σ_r in system Π is associated; the number stored in register r is encoded by the number of spikes in neuron σ_r. Specifically, if register r stores number $n \geq 0$, then there are $5n$ spikes in neuron σ_r. For each label l_i of an instruction in M, a neuron σ_{l_i} is associated. During the simulation, when neuron σ_{l_i} receives 6 spikes, it becomes active and starts to simulate instruction l_i: (OP(r), l_j, l_k) of M: the

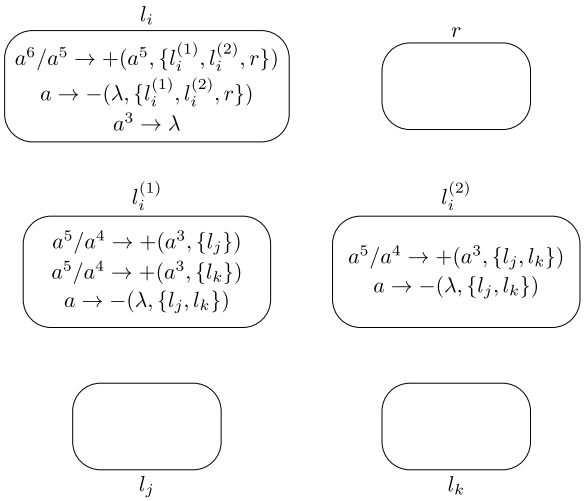

Figure 1. Module ADD simulating the ADD instruction.

process starts with neuron σ_{l_i} activated, operates on the number of spikes in neuron σ_r as requested by OP, then sends 6 spikes into neuron σ_{l_j} or σ_{l_k}, which becomes active in this way. Since there is no initial synapse in system Π, some synapses are created to pass spikes to target neurons with synapse creation rules, after that the created synapses will be deleted when simulation completes by synapse deletion rules. When neuron σ_{l_h} (associated with the halting instruction l_h of M) is activated, a computation in M is completely simulated by system Π.

The following describes the works of ADD, SUB, and FIN modules of the SN P systems with self-organization.

Module ADD (shown in Fig. 1): Simulating the ADD instruction l_i: (ADD(r), l_j, l_k).

Initially, there is no synapse in system Π, and all the neurons have no spike with exception that neuron σ_{l_0} has 6 spikes. This means system Π starts by simulating initial instruction l_0. Let us assume that at step t, an instruction l_i: (ADD(r), l_j, l_k) has to be simulated, with 6 spikes present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neurons, except in those neurons associated with registers.

At step t, neuron σ_{l_i} has 6 spikes, and synapse creation rule $a^6/a^5 \to +(a^5, \{l_i^{(1)}, l_i^{(2)}, r\})$ is applied in σ_{l_i}, it generates three synapses connecting neuron σ_{l_i} to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and σ_r. Meanwhile, it consumes 5 spikes (one spike remaining) and sends 5 spikes to each of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and σ_r. The number of spikes in neuron σ_r is increased by 5, which simulates adding 1 to register r of M. At step $t+1$, neuron σ_{l_i} deletes the three synapses created at step t by using rule $a \to -(\lambda, \{l_i^{(1)}, l_i^{(2)}, r\})$. At the same moment, neuron $\sigma_{l_i^{(2)}}$ uses synapse creation rule $a^5/a^4 \to +(a^3, \{l_j, l_k\})$, and creates two synapses to neurons σ_{l_j} and σ_{l_k}, as well as sends 3 spikes to each of the two neurons. At step $t+2$, neuron $\sigma_{l_i^{(2)}}$ deletes the two synapses by using synapse deletion rule $a \to -(\lambda, \{l_j, l_k\})$. In neuron $\sigma_{l_i^{(1)}}$, there are 5 spikes at step $t+1$ such that both of synapse creation rules $a^5/a^4 \to +(a^3, \{l_j\})$ and $a^5/a^4 \to +(a^3, \{l_k\})$ are enabled, but only one of them is non-deterministically used.

- If rule $a^5/a^4 \to +(a^3, \{l_j\})$ is chosen to use, neuron $\sigma_{l_i^{(1)}}$ creates a synapse and sends 3 spikes to neuron σ_{l_j}. In this case, neuron σ_{l_j} accumulates 6 spikes, which means system Π starts to simulate instruction l_j of M. One step later, with one spike inside neuron $\sigma_{l_i^{(1)}}$ uses rule $a \to -(\lambda, \{l_j, l_k\})$ to delete the synapse to neuron σ_{l_j}, and neuron σ_{l_k} removes the 3 spikes (from neuron $\sigma_{l_i^{(2)}}$) by the forgetting rule $a^3 \to \lambda$.
- If rule $a^5/a^4 \to +(a^3, \{l_k\})$ is selected to apply, neuron $\sigma_{l_i^{(1)}}$ creates a synapse and sends 3 spikes to neuron σ_{l_k}. Neuron σ_{l_k} accumulates 6 spikes, which indicates system Π goes to simulate instruction l_k of M. One step later, neuron σ_{l_j} removes the 3 spikes by using forgetting rule $a^3 \to \lambda$, and the synapse from neuron $\sigma_{l_i^{(1)}}$ to σ_{l_k} is deleted by using rule $a \to -(\lambda, \{l_j, l_k\})$ in neuron $\sigma_{l_i^{(1)}}$.

Therefore, from firing neuron σ_{l_i}, system Π adds 5 spikes to neuron σ_r and non-deterministically activates one of the neurons σ_{l_j} and σ_{l_k}, which correctly simulates the ADD instruction l_i: (ADD(r), l_j, l_k). When the simulation of ADD instruction is completed, the ADD module returns to its initial topological structure, i.e., there is no synapse in the module. The dynamic transformation of topological structure and the numbers of spikes in neurons of ADD module during the ADD instruction simulation with neuron σ_{l_j} or σ_{l_k} finally activated is shown in Figs 2 and 3. In the figures, the spiking rules are omitted for clear illustration, neurons are represented by circles with the number of spikes and directed edges is used to represent the synapses.

Module SUB (shown in Fig. 4): Simulating the SUB instruction l_i: (SUB(r), l_j, l_k).

Given starting time stamp t, system Π simulates a SUB instruction l_i: (SUB(r), l_j, l_k). Let s_r be the number of SUB instructions acting on register r and the set of labels of instructions acting on register r be $\{i_1, i_2, \ldots, i_{s_r}\}$. Obviously, it holds $i \in \{i_1, i_2, \ldots, i_{s_r}\}$.

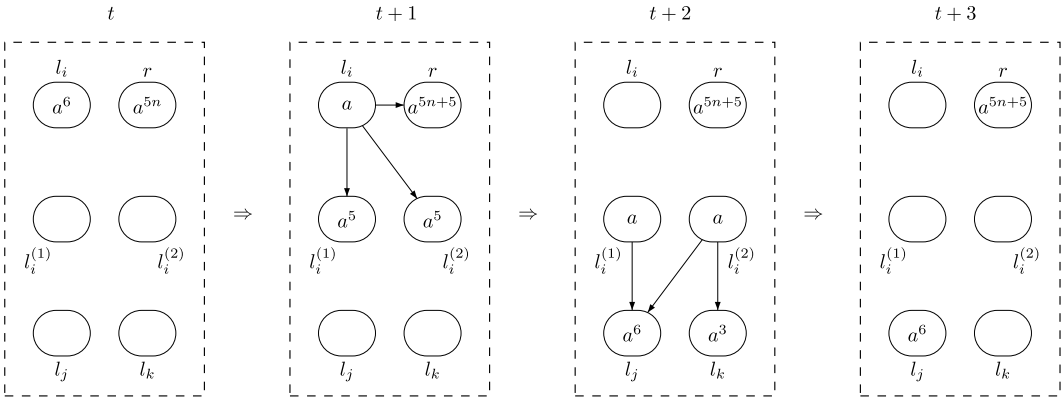

Figure 2. The dynamic transformation of topological structure and the numbers of spikes in neurons of ADD module during the ADD instruction simulation with neuron σ_{l_j} finally activated.

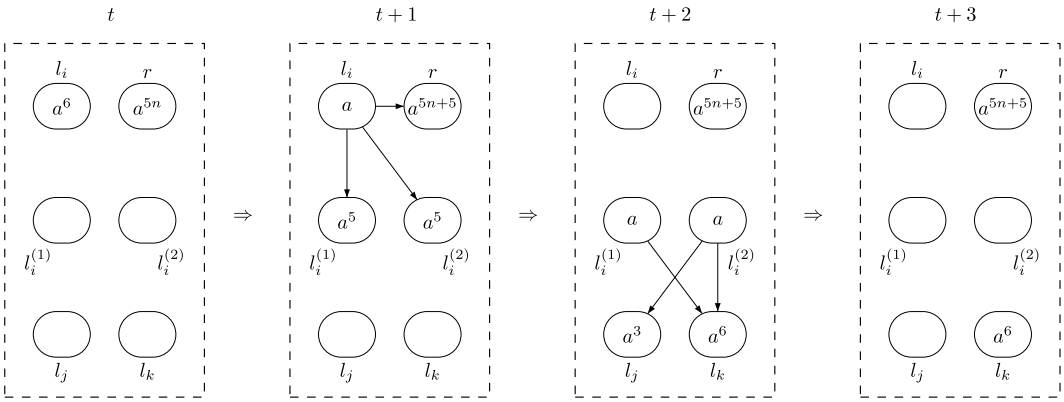

Figure 3. The dynamic transformation of topological structure and the numbers of spikes in neurons of ADD module during the ADD instruction simulation with neuron σ_{l_k} finally activated.

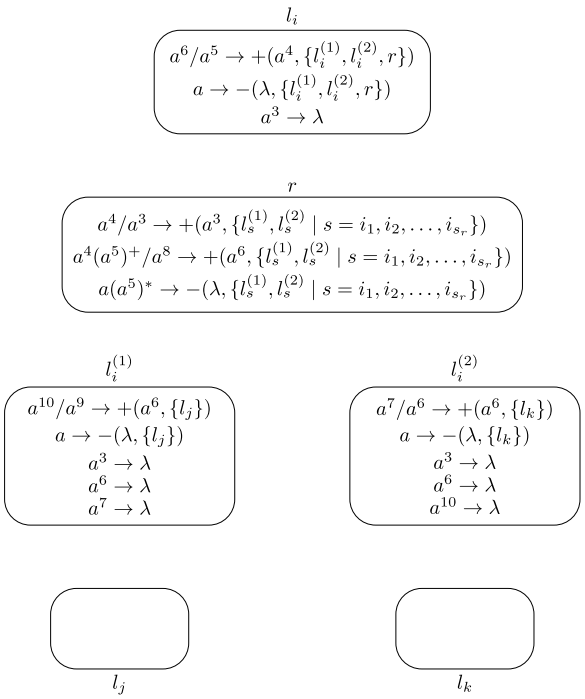

Figure 4. Module SUB simulating the SUB instruction.

At step t, neuron σ_{l_i} has 6 spikes, and becomes active by using synapse creation rule $a^6/a^5 \to +(a^4, \{l_i^{(1)}, l_i^{(2)}, r\})$, creating synapses and sending 4 spikes to each of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and σ_r. With 4 spikes inside, neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ keep inactive at step $t+1$ because no rule can be used. In neuron σ_r, it has the following two cases.

- If neuron σ_r has $5n$ ($n>0$) spikes (corresponding to the fact that the number stored in register r is n, and $n>0$), then by receiving 4 spikes from neuron σ_{l_i}, it accumulates $5n+4$ spikes and becomes active by using rule $a^4(a^5)^+/a^8 \to +(a^6, \{l_s^{(1)}, l_s^{(2)} | s = i_1, i_2, \ldots, i_{s_r}\})$ at step $t+1$. It creates a synapse to each of neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$ with $s = i_1, i_2, \ldots, i_{s_r}$ and sending 6 spikes to the neurons. By consuming 8 spikes, the number of spikes in neuron σ_r becomes $5n+4-8 = 5(n-1)+1$ $(n \geq 0)$ such that rule $a(a^5)^* \to -(\lambda, \{l_s^{(1)}, l_s^{(2)} | s = i_1, i_2, \ldots, i_{s_r}\})$ is enabled and applied at step $t+2$. With application of the rule, neuron σ_r removes the synapses from neuron σ_r to neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$, $s = i_1, i_2, \ldots, i_{s_r}$. Meanwhile, neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$ with $s \neq i$ remove the 6 spikes by using forgetting rule $a^6 \to \lambda$, and neuron $\sigma_{l_i^{(2)}}$ removes the 10 spikes by forgetting rule $a^{10} \to \lambda$. Neuron $\sigma_{l_i^{(1)}}$ accumulates 10 spikes (4 spikes from neuron σ_{l_i} and 6 spikes from neuron σ_r), and rule $a^{10}/a^9 \to +(a^6, \{l_j\})$ is applied at step $t+2$, creating a synapse and sending 6 spikes to neuron σ_{l_j}. In this case, neuron σ_{l_j} receives 6 spikes, which means system Π starts to simulate instruction l_j of M. One step later, the synapse from neuron $\sigma_{l_i^{(1)}}$ to neuron σ_{l_j} is deleted by using synapse deletion rule $a \to -(\lambda, \{l_j\})$.
- If neuron σ_r has no spike (corresponding to the fact that the number stored in register r is 0), then after receiving 4 spikes from neuron σ_{l_i}, it has 4 spikes and rule $a^4/a^3 \to +(a^3, \{l_s^{(1)}, l_s^{(2)} | s = i_1, i_2, \ldots, i_{s_r}\})$ is used, creating a synapse to each of neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$ with $s = i_1, i_2, \ldots, i_{s_r}$ and sending 3 spikes to the neurons. Neuron σ_r remains one spike, and synapse deletion rule $a(a^5)^* \to -(\lambda, \{l_s^{(1)}, l_s^{(2)} | s = i_1, i_2, \ldots, i_{s_r}\})$ is applied at step $t+2$, removing the synapses from neuron σ_r to neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$, $s = i_1, i_2, \ldots, i_{s_r}$. At the same moment, neurons $\sigma_{l_s^{(1)}}$ and $\sigma_{l_s^{(2)}}$ with $s \neq i$ remove the 3 spikes by using forgetting rule $a^3 \to \lambda$, and neuron $\sigma_{l_i^{(1)}}$ removes 7 spikes using spiking rule $a^7 \to \lambda$. Having 7 spikes, Neuron $\sigma_{l_i^{(2)}}$ becomes active by using rule $a^7/a^6 \to +(a^6, \{l_k\})$ at step $t+2$, creating a synapse to neuron σ_{l_k} and sending 6 spikes to neuron σ_{l_k}. In this case, neuron σ_{l_k} receives 6 spikes, which means system Π starts to simulate instruction l_k of M. At step $t+3$, neuron $\sigma_{l_i^{(2)}}$ uses rule $a \to -(\lambda, \{l_k\})$ to remove the synapse to neuron σ_{l_k}.

The simulation of SUB instruction performs correctly: System Π starts from σ_{l_i} having 6 spikes and becoming active, and ends in neuron σ_{l_j} receiving 6 spikes (if the number stored in register r is great than 0 and decreased by one), or in neuron σ_{l_k} receiving 6 spikes (if the number stored in register r is 0).

When the simulation of SUB instruction is completed, the SUB module returns to its initial topological structure, i.e., there is no synapse in the module. The dynamic transformation of topological structure and the numbers of spikes in involved neurons in the SUB instruction simulation with neuron σ_{l_j} (resp. neuron σ_{l_k}) finally activated is shown in Fig. 5 (resp. Fig. 6).

Module FIN (shown in Fig. 7) – outputting the result of computation.

Assume that at step t the computation in M halts, i.e., the halting instruction is reached. In this case, neuron σ_{l_h} in Π receives 6 spikes. At that moment, neuron σ_1 contains $5n$ spikes, for the number $n \geq 1$ stored in register 1 of M. With 6 spikes inside, neuron σ_{l_h} becomes active by using rule $a^6/a^5 \to +(a^2, \{1\})$, creating a synapse to neuron σ_1 and sending 2 spikes to neuron σ_1. Neuron σ_{l_h} ends with one spike, and rule $a \to -(\lambda, \{1\})$ is used, removing the synapse to neuron σ_1 one step later.

After neuron σ_1 receives the 2 spikes from neuron σ_{l_h}, the number of spikes in neuron σ_1 becomes $5n+2$ and rule $a^2(a^5)^+/a \to +(\lambda, \{0\})$ is enabled and applied at step $t+2$. By using the rule, neuron σ_1 consumes one spike and creates a synapse to the environment. Neuron σ_1 contains $5n+1$ spikes such that spiking rule $a(a^5)^+/a^5 \to a$ is used, consuming 5 spikes and emitting one spike to the environment at step $t+3$. Note that the number of spikes in neuron σ_1 becomes $5(n-1)+1$. So, if the number of spikes in neuron σ_1 is not one, then neuron σ_1 will fire again in the next step sending one spike into the environment. In this way, neuron σ_1 can fire for n times, i.e., until the number of spikes in neuron σ_1 reaches one. For each time when neuron σ_1 fires, it sends one spike into the environment. So, in total, neuron σ_1 sends n spikes into the environment, which is exactly the number stored in register 1 of M at the moment when the computation of M halts. When neuron σ_1 has one spike, rule $a \to -(\lambda, \{0\})$ is used to remove the synapse from neuron σ_1 to the environment, and system Π eventually halts.

The dynamic transformation of topological structure of the FIN module and the numbers of spikes in the neurons of FIN module and in the environment are shown in Fig. 8.

Based on the description of the work of system Π above, the register machine M is correctly simulated by system Π, i.e., $N(M) = N_{all}(\Pi)$. We can check that each neuron in system Π has at most three rules, and no limit is imposed on the numbers of neurons and the synapses that can be created (or deleted) by using one synapse creation (or deletion) rule. Therefore, it concludes $N_* SPSO_{all}(cre_*, del_*, rule_5) = NRE$.

This concludes the proof.

As number acceptor. Register machine can work in the accepting mode. Number n is accepted by register machine M' as follows. Initially, number n is stored in the first register of M' and all the other registers are empty. If the computation starting in this configuration eventually halts, then the number n is said to be accepted by register machine M'. The set of numbers accepted by register machine M' is denoted by $N_{acc}(M')$. It is known

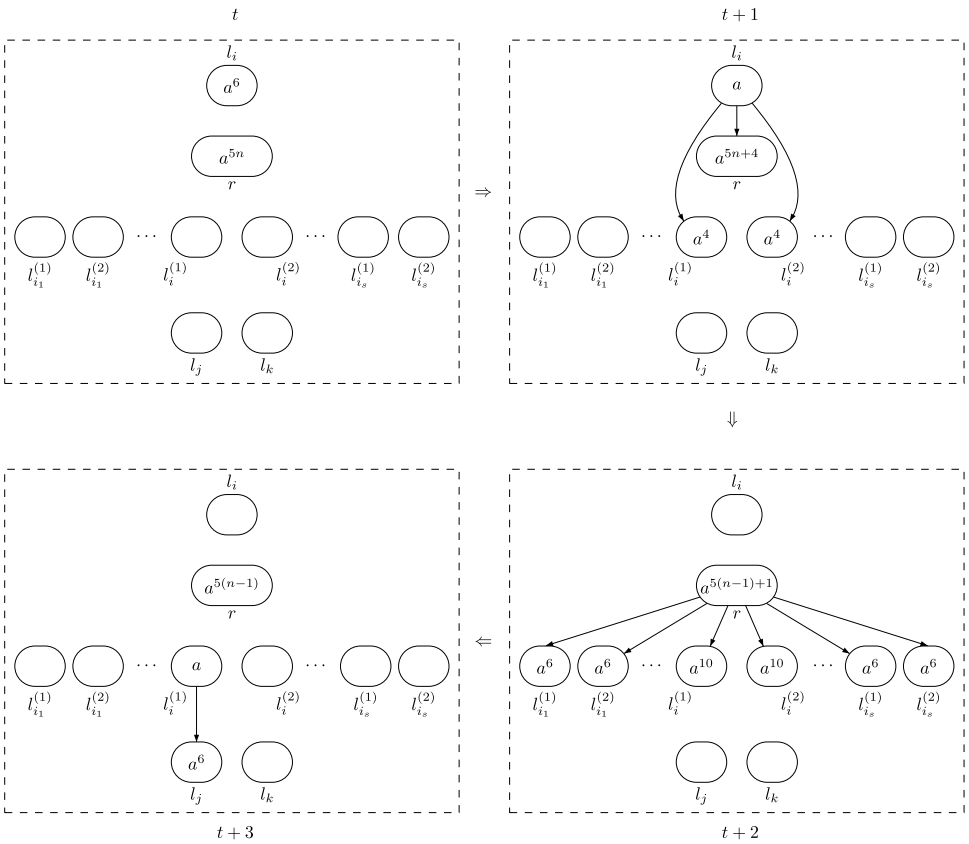

Figure 5. The dynamic transformation of topological structure and the numbers of spikes in involved neurons in the SUB instruction simulation with neuron σ_{l_j} finally activated.

that all the sets of numbers in *NRE* can be accepted by register machine M', even using the deterministic register machine; i.e. the machine with the ADD instructions of the form l_i: (ADD(r), l_j, l_k) where $l_j = l_k$ (in this case, the instruction is written in the form l_i: (ADD(r), l_j))[44].

Theorem 5. $N_*SPSO_{acc}(cre_*, del_*, rule_5) = NRE$.

Proof. It only has to prove $NRE \subseteq N_*SPSO_{acc}$ (cre_*, del_*, $rule_5$), since the converse inclusion is straightforward from the Turing-Church thesis. In what follows, an SN P system Π' with self-organization working in accepting mode is constructed to simulate a deterministic register machine $M' = (m, H, l_0, l_h, I)$ working in the acceptive mode. Actually, the proof is given by modifying the proof of Theorem 4.

Each register r of M' is associated with a neuron σ_r in system Π', and for each instruction l_i of M' a neuron σ_{l_i} is associated. A number n stored in register r is represented by $5n$ spikes in neuron σ_r.

The system Π' consists of an INPUT module, deterministic ADD and SUB modules. The INPUT module is shown in Fig. 9, where all neurons are initially empty with the exception that input neuron σ_{in} has 8 spikes. Spike train $10^{n-1}1$ is introduced into the system through input neuron σ_{in}, where the internal between the two spikes in the spike train is $(n+1)-1=n$, which indicates that number n is going to be accepted by system Π'.

Assuming at step t neuron σ_{in} receives the first spike. At step $t+1$, neuron σ_{in} contains 9 spikes, and rule $a^9/a^6 \to +(a^6, \{I_1, I_2\})$ is used, creating a synapse from neuron σ_{in} to neurons σ_{I_1} and σ_{I_2}. Meanwhile, neuron σ_{in} sends 6 spikes to the two neurons. In neuron σ_{in}, 6 spikes are consumed and 3 spikes remain. With 6 spikes inside, neurons σ_{I_1} and σ_{I_2} become active at step $t+2$. Neuron σ_{I_1} uses rule $a^6/a \to +(\lambda, \{I_2\})$ to create a synapse to neuron σ_{I_2}; and neuron σ_{I_2} uses rule $a^6/a \to +(\lambda, \{I_1, 1\})$ to create a synapse to each of neurons σ_{I_1} and σ_1. Each of neurons σ_{I_1} and σ_{I_2} has 5 spikes left. From step $t+3$ on, neurons σ_{I_1} and σ_{I_2} fire and begin to exchange 5 spikes between them. In this way, neuron σ_1 receives 5 spikes from neurons σ_{I_2} at each step.

At step $t+n$, neuron σ_{in} receives the second spike from the environment, accumulating 4 spikes inside. At step $t+n+1$, neuron σ_{in} fires for the second time by using spiking rule $a^4/a^3 \to a^3$, sending 3 spikes to neurons σ_{I_1} and σ_{I_2}. Each of neurons σ_{I_1} and σ_{I_2} accumulates 8 spikes. At step $t+n+2$, neuron σ_{I_1} uses synapse creation rule $a^8/a^6 \to +(a^6, \{l_0\})$, creating a synapse to neuron σ_{l_0} and sending 6 spikes to neuron σ_{l_0}. This means that system Π' starts to simulate the initial instruction l_0 of register machine M'. Meanwhile, neuron σ_{I_2} uses synapse deletion rule $a^8/a \to -(\lambda, \{I_1\})$, removing the synapse from neuron .. to neuron σ_{I_1}. In the next step, neuron σ_{I_1} creates a synapse to neuron σ_1 and sends 5 spikes to neuron σ_1 by using rule $a^7 \to +(a^5, \{1\})$.

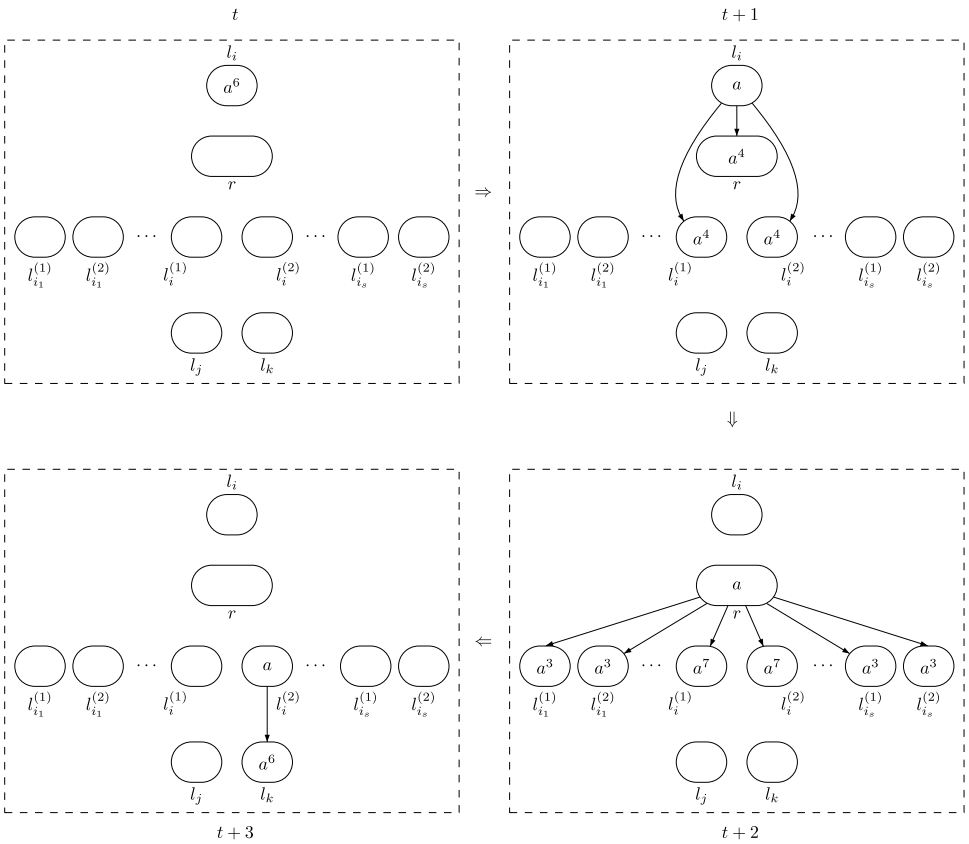

Figure 6. The dynamic transformation of topological structure and the numbers of spikes in involved neurons in the SUB instruction simulation with neuron σ_{l_k} finally activated.

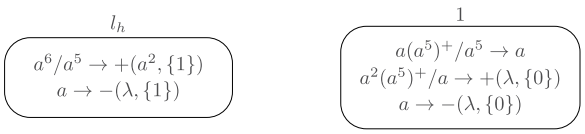

Figure 7. Outputting the computation result.

From step $t+3$ to $t+n+1$, neuron σ_1 receives 5 spikes in each step from neuron σ_{I_2}, thus in total accumulating $5(n-1)$ spikes. Neuron σ_1 receives no spike at step $t+n+2$, and gets 5 spikes from neuron σ_{I_2} at step $t+n+3$. After that, no more spikes are sent to neuron σ_{I_2}. Neuron σ_1 contains $5n$ spikes, which indicates the number to be accepted by register machine M' is n. At step $t+n+4$, neuron σ_{I_2} uses rule $a^2 \to -(\lambda, \{1\})$, deleting the synapse to neuron σ_1.

The dynamic transformation of topological structure of INPUT module and the numbers of spikes in the neurons of INPUT module are shown in Fig. 10.

The deterministic ADD module is shown in Fig. 11, whose function is rather clear. By receiving 6 spikes, neuron σ_{l_i} becomes active, creating a synapse and sending 5 spikes to each of neurons σ_r, $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. The number of spikes in neuron σ_r is increased by 5, which simulates the number stored in register 1 is increased by one. In the next step, neuron σ_{l_i} uses rule $a \to -(\lambda,\ \{l_i^{(1)},\ l_i^{(2)},\ r\})$, removing the synapses from neuron σ_{l_i} to neurons σ_r, $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, there are 5 spikes. The two neurons become active by using rule $a^5/a^4 \to +(a^3, \{l_j\})$. Each of them creates a synapse to neuron σ_{l_j} and emits 3 spikes to neuron σ_{l_j}. In this way, neuron σ_{l_j} accumulates 6 spikes inside, which means the system Π' goes to simulate instruction l_j of M'. The synapses from neuron $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ to neuron σ_{l_j} will be removed by using synapse deletion rule $a \to -(\lambda, \{l_j\})$ in neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$.

Module SUB remains unchanged, as shown in Fig. 4. Module FIN is removed, with neuron σ_{l_h} remaining in the system, but having no rule inside. When neuron σ_{l_h} receives 6 spikes, it means that the computation of register machine M' reaches instruction l_h and stops. Having 6 spikes inside, neuron σ_{l_h} cannot become active for no rule can be used. In this way, the work of system Π' halts.

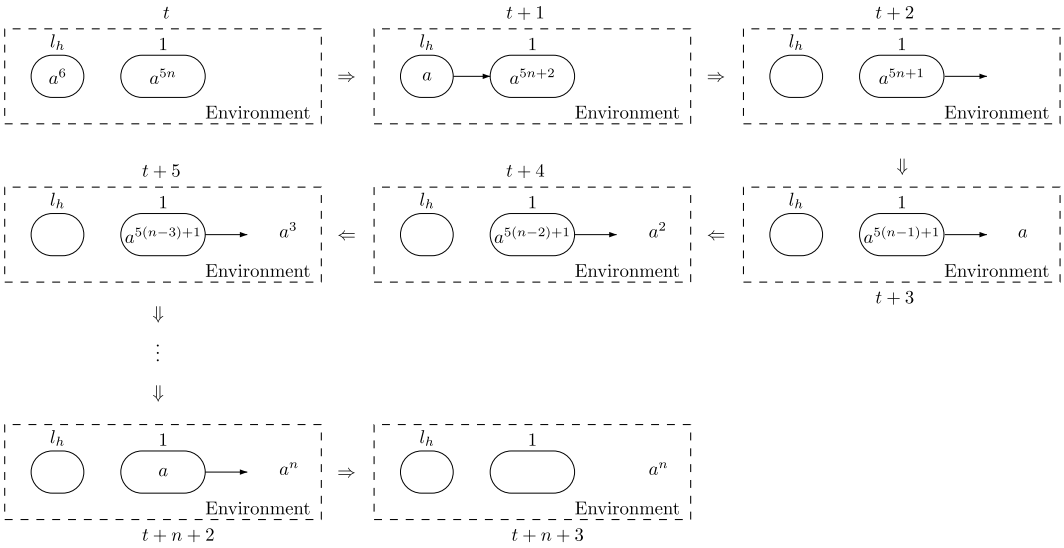

Figure 8. The dynamic transformation of topological structure of the FIN module and the numbers of spikes in the neurons of FIN module and the environment.

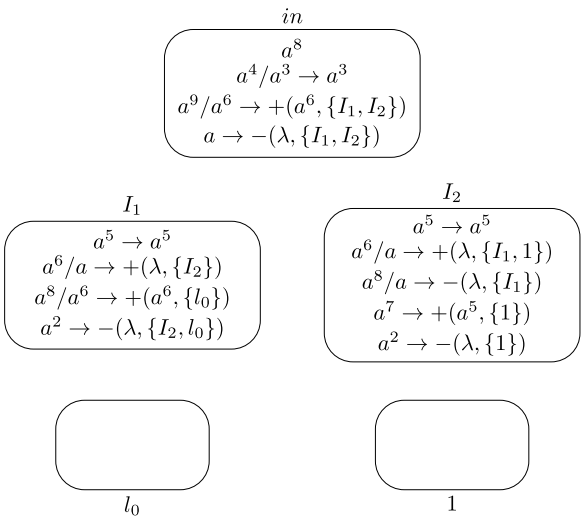

Figure 9. The INPUT module of system Π′.

Based on the description of the implementation of system Π′ above, it is clear that the register machine M' in acceptive mode is correctly simulated by the system Π′ working in acceptive mode, i.e., $N_{acc}(M') = N_{acc}(\Pi')$.

We can check that each neuron in system Π′ has at most five rules, and no limit is imposed on the numbers of neurons and the synapses that can be created (or deleted) with using one synapse creation (or deletion) rule. Therefore, it concludes $N_*SPSO_{acc}(cre_*, del_*, rule_5) = NRE$.

As function computing device. A register machine M can compute a function $f: \mathbf{N}^k \to \mathbf{N}$ as follows: the arguments are introduced in special registers $r_1, r_2, \ldots, r_k$ (without loss of the generality, it is assumed that the first k registers are used). The computation starts with the initial instruction l_0. if the register machine halts, i.e., reaches HALT instruction l_h, the value of the function is placed in another specified register, labelled by r_t, with all registers different from r_t storing number 0. The partial function computed by a register machine M in this way is denoted by $M(n_1, n_2, \ldots, n_k)$. All Turing computable functions can be computed by register machine in this way.

Several universal register machines for computing functions were defined. 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 addressed by Minsky, universal register machine can compute any $\varphi_x(y)$ by inputting a couple of numbers $g(x)$ and y in registers 1 and 2, and the result can be obtained in register 0[47].

In the following proof of universality, a specific universal register machine M_u from[47] is used, the machine $M_u = (8, H, l_0, l_h, I)$ presented in Fig. 12. In this universal register machine M_u, there are 8 registers (numbered from 0 to 7) and 23 instructions, and the last instruction is the halting one. As described above, the input numbers

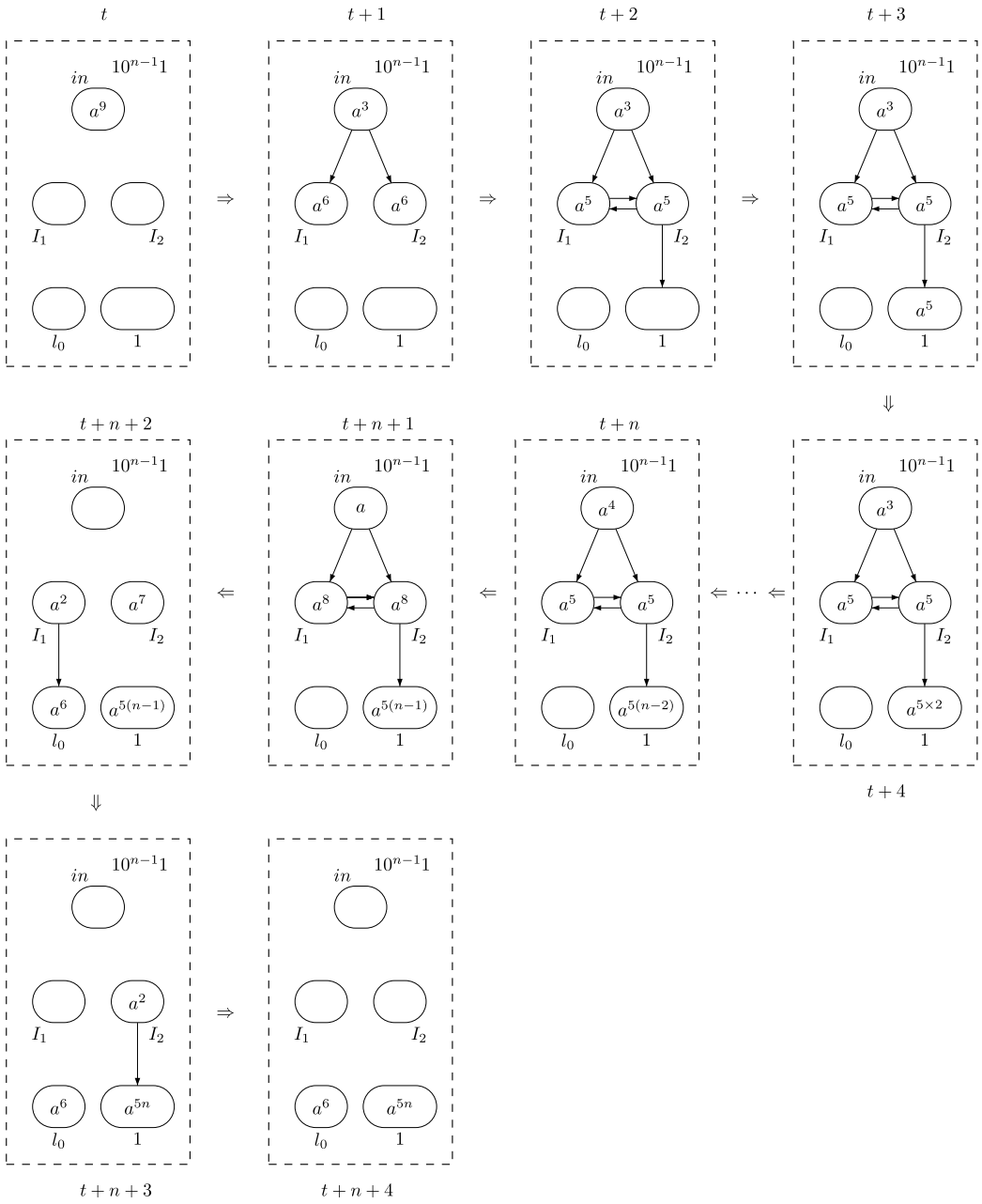

Figure 10. The dynamic transformation of topological structure of INPUT module and the numbers of spikes in the neurons of INPUT module.

(the "code" of the partial recursive function to compute and the argument for this function) are introduced in registers 1 and 2, and the result is outputted in register 0 when the machine M_u halts.

A modification is necessary to be made in M_u, because the subtraction operation in the register where the result is placed is not allowed in the construction of the previous Theorems, but register 0 of M_u is subject of such operations. That is why an extra register is needed - labeled with 8 - and the halt instruction l_h of M_u should be replaced by the following instructions:

$$l_{22}: (\texttt{SUB}(0),\ l_{23},\ l_h'),\ l_{23}: (\texttt{ADD}(8),\ l_{22}),\ l_h': \texttt{HALT}.$$

Therefore, the modified universal register machine M_u' has 9 registers, 24 ADD and SUB instructions, and 25 labels. The result of a computation of M_u' is stored in register 8

Theorem 6 *There is a universal SN P system with self-organization having 87 neurons for computing functions.*

Proof. An SN P system with self-organization Π'' is constructed to simulate the computation of the universal register machine M_u'. Specifically, the system Π'' consists of deterministic ADD modules, SUB modules, as well as

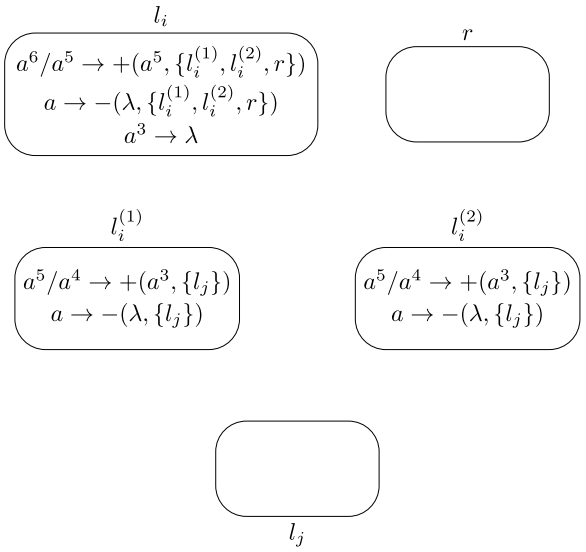

Figure 11. The deterministic ADD module of system Π′.

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

Figure 12. The universal register machine M_u from[47].

an INPUT module and an OUTPUT module. The deterministic ADD module shown in Fig. 11 and SUB module shown in Fig. 4 can be used here to simulate the deterministic ADD instruction and SUB instruction of M'_u. The INPUT module introduces the necessary spikes into the system by reading a spike train from the environment, and the OUTPUT module outputs the computation result.

With each register r of M'_u, a neuron σ_r in system Π'' is associated; the number stored in register r is encoded by the number of spikes in neuron σ_r. If register r holds the number $n \geq 0$, then neuron σ_r contains $5n$ spikes. With each instruction l_i in .., a neuron σ_{l_i} in system Π'' is associated. If neuron σ_{l_i} has 6 spikes inside, it becomes active and starts to simulate the instruction l_i. When neuron $\sigma_{l'_h}$ (associated with the label l'_h of the halting instruction of M'_u) receives 6 spikes, the computation in M'_u is completely simulated by the system Π''; the number of spikes emitted into the environment from the output neuron, i.e., neuron σ_8, corresponds to the result computed by M'_u (stored in register 8).

The tasks of loading $5g(x)$ spikes in neuron σ_1 and $5y$ spikes in neuron σ_2 by reading the spike train $10^{g(x)-1}$ $10^{y-1}1$ through input neuron σ_{in} can be carried out by the INPUT module shown in Fig. 13.

Initially, all the neurons contain no spike inside, with the exception that neuron σ_{in} has 15 spikes. It is assumed at step t neuron σ_{in} reads the first spike from the environment. With 16 spikes inside, neuron σ_{in} becomes active by using rule $a^{16}/a^6 \to +(a^6, \{I_1, I_2\})$ at step $t+1$. It creates a synapse and sends 6 spikes to each of neurons σ_{I_1} and σ_{I_2}. Subsequently, neuron σ_{in} keeps inactive (for no rule can be used) until the second spike arrives at step $t+g(x)$.

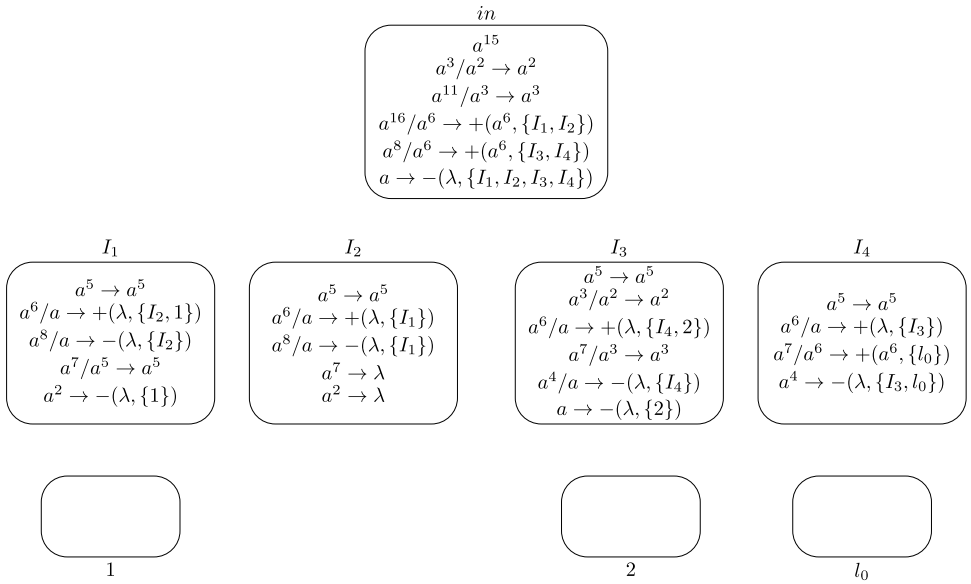

Figure 13. The INPUT module of system Π″.

Neuron σ_{I_1} has 6 spikes and uses rule $a^6/a \to +(\lambda, \{I_2, 1\})$ at step $t+2$ it creates a synapse to neurons σ_{I_2} and σ_1 and sends 5 spikes to each of the two neurons. Meanwhile, neuron σ_{I_2} creates a synapse to neuron σ_{I_2} and sends 5 spikes to it. From step $t+3$ on, neuron σ_{I_1} sends 5 spikes to neuron σ_{I_1} and exchanges 5 spikes with neuron σ_{I_2} in each step.

At step $t+g(x)$, neuron σ_{in} receives the second spike from the environment. By then it accumulates 11 spikes inside. At step $t+g(x)+1$, neuron σ_{in} fires by using spiking rule $a^{11}/a^3 \to a^3$, and sends 3 spikes to neurons σ_{I_1} and σ_{I_2}. Each of neurons σ_{I_1} and .. contains 8 spike, which will be remained in σ_{in}.

At step $t+g(x)+2$, neuron σ_{I_1} applies synapse deletion rule $a^8/a \to -(\lambda, \{I_1\})$ and removes the synapse to neuron σ_{I_0}, meanwhile neuron σ_{I_2} removes the synapse to neuron σ_{I_1}. The two neurons stop to exchange spikes with each other. At step $t+g(x)+3$, neuron σ_{I_1} has 7 spikes and fires by using spiking rule $a^7/a^5 \to a^5$, and sends 5 spikes to neuron σ_1. In the next step, neuron σ_{I_1} removes the synapse to neuron σ_1, and cannot send spikes to neuron σ_1.

In general, in each step from step $t+3$ to $t+g(x)+1$, neuron σ_1 receives 5 spikes from neuron σ_{I_1}, in total receiving $5(g(x)-1)$ spikes; at step $t+g(x)+2$, no spike arriving in neuron σ_1, and at step $t+g(x)+3$, 5 spikes reaching neuron σ_1. In this way, neuron σ_1 accumulates $5g(x)$ spikes, which simulates number $g(x)$ is stored in register 1 of M'_u.

At step $t+g(x)+2$, neuron σ_{in} contains 8 spikes such that rule $a^8/a^6 \to +(a^6, \{I_3, I_4\})$ is used, creating a synapse and sends 6 spikes to each of neurons σ_{I_3} and σ_{I_4}. At step $t+g(x)+3$, neurons σ_{I_3} and σ_{I_4} create synapses to each other, meanwhile neuron σ_{I_3} creates a synapse to neuron σ_2. From step $t+g(x)+4$ on, neuron σ_{I_3} begins to exchange 5 spikes with σ_{I_4} and send 5 spikes to neuron σ_2 in each step. At step $t+g(x)+y$, neuron σ_{in} receives the third spike from the environment, accumulating 3 spikes inside. One step later, it fires by using spiking rule $a^3/a^2 \to a^2$, sending 2 spikes to neurons σ_{I_1}, σ_{I_2}, σ_{I_3} and σ_{I_4}. With 2 spikes inside, neuron σ_1 removes its synapse to neuron σ_1 by rule $a^2 \to -(\lambda, \{1\})$; while neuron σ_{I_2} forgets the two spikes by forgetting rule $a^2 \to \lambda$. By receiving 2 spikes from neuron σ_{in}, neurons σ_{I_3} and σ_{I_4} contain 7 spikes. At step $t+g(x)+y+2$, neuron σ_{I_3} fires by using rule $a^7/a^3 \to a^3$ and sends 3 spikes to neurons σ_2 and σ_{I_4}. The number of spikes in neuron σ_2 is $5(y-1)+3$. Neuron σ_{I_4} consumes 6 spikes, creates a synapse and sends 6 spikes to neuron σ_{l_0}. This means system Π" starts to simulate the initial instruction l_0 of M'_u.

At step $t+g(x)+y+3$, neuron σ_{I_3} has 4 spikes, and it becomes active by rule $a^4/a \to -(\lambda, \{I_4\})$, and removes its synapse to neuron σ_{I_3} and ends with 3 spikes. In the next step, neuron σ_{I_3} fires by using spiking rule $a^3/a^2 \to a^2$, emitting 2 spikes to neuron σ_2. In this way, the number of spikes in neuron σ_2 becomes $5(y-1)+3+2=5y$, which indicates number y is stored in register 2 of M'_u.

The deterministic ADD module shown in Fig. 11 and SUB module shown in Fig. 4 can be used to simulate ADD and SUB instructions of M'_u. The FIN module shown in Fig. 7 can be used to output the computation result with changing neuron σ_1 into σ_8.

Until now, we have used

- 9 neurons for 9 registers,
- 25 neurons for 25 labels,
- 20 neurons for 10 ADD instructions,
- 28 neurons for 14 SUB instruction,
- 5 additional neurons in the INPUT module,

which comes to a total of 87 neurons.
This concludes the proof.

Discussion and Future Works

In this work, a novel variant of SN P systems, namely SN P systems with self-organization, is introduced. As results, it is proven that the systems are Turing universal, i.e., they can compute and accept the family of sets of Turing computable natural numbers. With 87 neurons, the system can compute any Turing computable recursive function, thus achieving Turing universality.

There has been a research focus on the construction of small universal SN P with less computing resource, i.e. less number of neuron in use[17,48–52]. It is of interest that whether we can reduce the number of neurons in universal SN P systems with self-organization as function computing devices. A possible way is to construct ADD-ADD, ADD-SUB and SUB-ADD modules to perform particular consecutive ADD-ADD, ADD-SUB, and SUB-ADD instructions of M_u'.

SN P systems with learning function/capabiliy is a promising direction. Learning strategies and feedback mechanism have been intensively studied and investigated in conventional artificial neural networks. It is worthy to look into these techniques and transplant these ideas into SN P systems with self-organization.

In research of using artificial neural networks to recognize digital English letters, database MNIST (Mixed National Institute of Standards and Technology database) is widely used for training various letter recognition systems[53], and for training and testing in the field of machine learning[54]. For further research, SN P systems with self-organization may be used to recognize handwritten digits letters and other possible pattern recognition problems. Since the data structure of SN P systems is binary sequences, an extra task of transmitting letters or pictures into binary sequences should be addressed. A possible way is transmitting digital numbers of pixels of pictures to binary form. Also, local binary pattern method, can be used to transmit pictures to binary forms.

Bioinformatics is s an interdisciplinary field that develops methods and software tools for understanding biological data[55]. Artificial intelligence based methods and data mining strategy have been used in processing biological data, see e.g.[56–62], it is worthy to processing biological data by SN P systems, such as DNA motif finding[63,64], nuclear export signal identification[65,66].

References

1. Chen, X., Perez-Jimenez, M. J., Valenciacabrera, L., Wang, B. & Zeng, X. Computing with viruses. *Theor. Comput. Sci.* **623,** 146–159 (2016).
2. Zhang, X., Tian, Y. & Jin, Y. A knee point-driven evolutionary algorithm for many-objective optimization. *IEEE T. Evolut. Compu.* **16,** 35–41 (2015).
3. Zhang, X., Tian, Y., Cheng, R. & Jin, Y. An efficient approach to nondominated sorting for evolutionary multiobjective optimization. *IEEE T. Evolut. Compu*. **19,** 201–213 (2015).
4. Gerstner, W. & Kistler, W. M. *Spiking neuron models: single neurons, populations, plasticity* (Cambridge university press, 2002).
5. Hagan, M. T., Demuth, H. B. & Beale, M. H. *Neural network design* (Pws Publishing Boston, 1996).
6. Ghosh-Dastidar, S. & Adeli, H. Spiking neural networks. *Int. J. Neural Syst.* **19,** 295–308 (2009).
7. Ionescu, M., Păun, Gh. & Yokomori, T. Spiking neural P systems. *Fund. Inform*. **71,** 279–308 (2006).
8. Maass, W. Networks of spiking neurons: the third generation of neural network models. *Neural Networks* **10,** 1659–1671 (1997).
9. Pan, L. & Păun, Gh. Spiking neural P systems: an improved normal form. *Theor. Comput. Sci.* **411,** 906–918 (2010).
10. Song, T. & Pan, L. Spiking neural P systems with rules on synapses working in maximum spiking strategy. *IEEE T. Nanobiosci.* **14,** 465–477 (2015).
11. Cavaliere, M. *et al.* Asynchronous spiking neural P systems. *Theor. Comput. Sci.* **410,** 2352–2364 (2009).
12. Song, T. & Pan, L. Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. *IEEE T. Nanobiosci.* **14,** 37–43 (2015).
13. Song, T., Pan, L. & Păun, Gh. Asynchronous spiking neural P systems with local synchronization. *Inform. Sciences* **219,** 197–207 (2012).
14. Păun, Gh. Spiking neural P systems with astrocyte-like control. *J. Univers. Comput. Sci.* **13,** 1707–1721 (2007).
15. Chen, H., Freund, R., Ionescu, M., Păun, Gh. & Pérez-Jiménez, M. J. On string languages generated by spiking neural P systems. *Fund. Inform*. **75,** 141–162 (2007).
16. Zeng, X., Xu, L., Liu, X. & Pan, L. On languages generated by spiking neural P systems with weights. *Inform. Sciences* **278,** 423–433 (2014).
17. Păun, A. & Păun, Gh. Small universal spiking neural P systems. *Biosystems* **90,** 48–60 (2007).
18. Song, T., Pan, L., Jiang, K., Song, B. & Chen, W. Normal forms for some classes of sequential spiking neural P systems. *IEEE T. Nanobiosci.* **12,** 255–264 (2013).
19. Pan, L. & Păun, Gh. Spiking neural P systems with anti-spikes. *Int. J. Comput. Commun.* **4,** 273–282 (2009).
20. Pan, L., Wang, J., Hoogeboom, H. J. & Pérez-Jiménez, M. J. Spiking neural P systems with weights. *Neural Comput.* **22,** 2615–2646 (2010).
21. Pan, L., Wang, J. & Hoogeboom, H. J. Spiking neural P systems with astrocytes. *Neural Comput.* **24,** 805–825 (2012).
22. Song, T., Wang, X., Zhang, Z. & Chen, Z. Homogenous spiking neural P systems with anti-spikes. *Neural Comput. Appl.* **24,** 1833–1841 (2013).
23. Zeng, X., Zhang, X. & Pan, L. Homogeneous spiking neural P systems. *Fund. Inform.* **97,** 275–294 (2009).
24. Zeng, X., Zhang, X., Song, T. & Pan, L. Spiking neural P systems with thresholds. *Neural Comput.* **26,** 1340–1361 (2014).
25. Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M. J. & Wang, T. Weighted fuzzy spiking neural P systems. *IEEE T. Fuzzy Syst*. **21,** 209–220 (2013).
26. Peng, H. *et al.* Fuzzy reasoning spiking neural P system for fault diagnosis. *Inform. Sciences* **235,** 106–116 (2013).
27. Ibarra, O. H., Păun, A. & Rodrguez-Patón, A. Sequential SNP systems based on min/max spike number. *Theor. Comput. Sci.* **410,** 2982–2991 (2009).
28. Song, T., Zou, Q., Li, X. & Zeng, X. Asynchronous spiking neural P systems with rules on synapses. *Neurocomputing* **151,** 1439–1445 (2015).
29. Cabarle, F. G. C., Adorna, H. N., Pérez-Jiménez, M. J. & Song, T. Spiking neural P systems with structural plasticity. *Neural Comput. Appl.* **26,** 1905–1917 (2015).
30. Ionescu, M. & Sburlan, D. *Several applications of spiking neural P systems*. Proceedings of the Fifth Brainstorming Week on Membrane Computing, Sevilla, Spain (2007).

31. Adl, A., Badr, A. & Farag, I. Towards a spiking neural P systems OS. *arXiv preprint arXiv:1012.0326* (2010).
32. Liu, X., Li, Z., Liu, J., Liu, L. & Zeng, X. Implementation of arithmetic operations with time-free spiking neural P systems. *IEEE T. Nanobiosci.* **14,** 617–624 (2015).
33. Zeng, X., Song, T., Zhang, X. & Pan, L. Performing four basic arithmetic operations with spiking neural P systems. *IEEE T. Nanobiosci.* **11,** 366–374 (2012).
34. Zhang, G., Rong, H., Neri, F. & Pérez-Jiménez, M. J. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *Int. J. Neural Syst.* **24**, 1440006 (2014).
35. Wang, T. *et al.* Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems *IEEE T. Power Syst*. **30** 1182–1194 (2015).
36. Păun, Gh., Rozenberg, G. & Salomaa, A. *The Oxford handbook of membrane computing* (Oxford University Press, Inc., 2010).
37. Maass, W. & Bishop, C. M. *Pulsed neural networks* (MIT press, 2001).
38. Siegelmann, H. T. & Sontag, E. D. On the computational power of neural nets. *J. Comput. Syst. Sci.* **50,** 132–150 (1995).
39. Buzsáki, G. Neural syntax: cell assemblies, synapsembles, and readers. *Neuron* **68,** 362–385 (2010).
40. Tetzlaff, C., Kolodziejski, C., Timme, M., Tsodyks, M. & Wörgötter, F. Synaptic scaling enables dynamically distinct short-and long-term memory formation. *BMC Neurosci.* **14,** P415 (2013).
41. Fritzke, B. Growing cell structures a self-organizing network for unsupervised and supervised learning. *Neural Networks* **7,** 1441–1460 (1994).
42. Ultsch, A. *Self-organizing neural networks for visualisation and classification* (Springer-Verlag, Berlin, 1993).
43. Gheorghe, M., Păun, Gh., Pérez-Jiménez, M. J. & Rozenberg, G. Spiking neural P systems, research frontiers of membrane computing: Open problems and research topics. *Int. J. Found. Comput. S.* **24,** 547–623 (2013).
44. Minsky, M. L. *Computation: finite and infinite machines* (Prentice-Hall, New Jersey, 1967).
45. Rozenberg, G. & Salomaa, A. *Handbook of formal languages*, vol. 3 (Springer-Verlag, Berlin, 1997).
46. Păun, Gh. *Membrane computing: an introduction* (Springer-Verlag, Berlin, 2002).
47. Korec, I. Small universal register machines. *Theor. Comput. Sci.* **168,** 267–301 (1996).
48. Neary, T. A universal spiking neural P system with 11 neurons. *Proceedings of the Eleventh International Conference on Membrane Computing*, Jena, Germany (2010).
49. Pan, L. & Zeng, X. A note on small universal spiking neural P systems. In *Lect. Notes Comput. Sci.* vol. 5957, 436–447 (Springer-Verlag, Berlin, 2010).
50. Păun, A. & Sidoroff, M. *Sequentiality induced by spike number in SNP systems: small universal machines*. Membrane Computing, 333–345 (Springer-Verlag, Berlin, 2012).
51. Song, T., Jiang, Y., Shi, X. & Zeng, X. Small universal spiking neural P systems with anti-spikes. *J. Comput. Theor. Nanos.* **10,** 999–1006 (2013).
52. Zhang, X., Zeng, X. & Pan, L. Smaller universal spiking neural P systems. In *Annales Societatis Mathematicae Polonae. Series 4: Fundamenta Informaticae*, **vol. 87,** 117–136 (2008).
53. Burges, C. J. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Disc.* **2,** 121–167 (1998).
54. Liu, C.-L., Nakashima, K., Sako, H. & Fujisawa, H. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern Recogn.* **36,** 2271–2285 (2003).
55. Bioinformatics-Wikipedia, the free encyclopedia., 5 May 2016.
56. Zou, Q., Li, J., Song, L., Zeng, X. & Wang, G. Similarity computation strategies in the microRNA disease network: a survey. *Brief Funct. Genomic* **15,** 55–64 (2015).
57. Lin, C. *et al.* LibD3C: Ensemble classifiers with a clustering and dynamic selection strategy. *Neurocomputing* **123,** 424–435 (2014).
58. Liu, B. *et al.* Pse-in-One: a web server for generating various modes of pseudo components of DNA, RNA, and protein sequences. *Nucleic Acids Res.* **W1,** W65–W71 (2015).
59. Zou, Q., Hu, Q., Guo, M. & Wang, G. Halign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy. *Bioinformatics* **31,** 2475–2481 (2015).
60. Zeng, J., Li, D., Wu, Y., Zou, Q. & Liu, X. An empirical study of features fusion techniques for protein-protein interaction prediction. *Curr. Bioinform.* **11,** 4–12 (2016).
61. Yunfeng, W. & S., K. Combining least-squares support vector machines for classification of biomedical signals: a case study with knee-joint vibroarthrographic signals. *J Exp. Theor. Artif. In.* **23,** 63–77 (2011).
62. Song, L. *et al.* ndna-prot: identification of DNA-binding proteins based on unbalanced classification. *BMC Bioinformatics* **15,** 1 (2014).
63. Liu, B., Liu, F., Fang, L., Wang, X. & Chou, K. repDNA: a python package to generate various modes of feature vectors for DNA sequences by incorporating user-defined physicochemical properties and sequence-order effects. *Bioinformatics* **31,** 1307–1309 (2015).
64. Zeng, X., Liao, Y., Liu, Y. & Zou, Q. Prediction and validation of disease genes using hetesim scores. *IEEE ACM T Comput. Bi.*, doi: 10.1109/TCBB.2016.2520947 (2016).
65. Zou, Q., Li, J., Song, L., Zeng, X. & Wang, G. Similarity computation strategies in the microrna-disease network: a survey. *BRIEF Funct. Genomic* **15,** 55–64 (2016).
66. Zeng, X., Zhang, X. & Zou, Q. Integrative approaches for predicting microrna function and prioritizing disease-related microRNA using biological interaction networks. *Brief Bioinform.* **17,** 193–203 (2015).

Acknowledgements

The research is under the auspices of National Natural Science Foundation of China (Nos 41276135, 31172010, 61272093, 61320106005, 61402187, 61502535, 61572522 and 61572523), Program for New Century Excellent Talents in University (NCET-13-1031), 863 Program (2015AA020925), and Fundamental Research Funds for the Central Universities (15CX05015A).

Author Contributions

T.S. and X.W. contributed the main idea, X.W. and F.G. performed the analysis and theoretical proofs, P.Z. and X.W. wrote the paper. All authors reviewed the manuscript.

Additional Information

Competing financial interests: The authors declare no competing financial interests.

How to cite this article: Wang, X. *et al.* On the Computational Power of Spiking Neural P Systems with Self-Organization. *Sci. Rep.* **6**, 27624; doi: 10.1038/srep27624 (2016).