Knowledge-Based Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Spiking neural P systems with inhibitory rules☆,☆☆

Hong Peng [a], Bo Li [a], Jun Wang [b,*], Xiaoxiao Song [b], Tao Wang [b], Luis Valencia-Cabrera [c], Ignacio Pérez-Hurtado [c], Agustín Riscos-Núñez [c], Mario J. Pérez-Jiménez [c]

[a] School of Computer and Software Engineering, Xihua University, Chengdu 610039, China
[b] School of Electrical Engineering and Electronic Information, Xihua University, Chengdu 610039, China
[c] Research Group of Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Sevilla 41012, Spain

ARTICLE INFO

Article history:
Received 1 November 2018
Received in revised form 20 September 2019
Accepted 22 September 2019
Available online xxxx

Keywords:
Membrane computing
Spiking neural P systems
Spiking neural P systems with inhibitory rules
Inhibitory synapse

ABSTRACT

Motivated by the mechanism of inhibitory synapses, a new kind of spiking neural P (SNP) system rules, called inhibitory rules, is introduced in this paper. Based on this, a new variant of SNP systems is proposed, called spiking neural P systems with inhibitory rules (SNP-IR systems). Different from the usual firing rules in SNP systems, the firing condition of an inhibitory rule not only depends on the state of the neuron associated with the rule but also is related to the states of other neurons. Moreover, from the perspective of topological structure, the new variant is shown as a directed graph with inhibitory arcs, and therefore seems to have more powerful control. The computational completeness of SNP-IR systems is discussed. In particular, it is proved that SNP-IR systems are Turing universal number accepting/generating devices. Moreover, we obtain a small universal function-computing device for SNP-IR systems consisting of 100 neurons.

1. Introduction

Membrane computing is a class of distributed parallel computing systems initiated by Gheorghe Păun [1], abstracted from the structure and functioning of biological cells as well as the cooperation of cell populations in tissues, organs, and biological neural networks [2]. These computing systems are known as P systems or membrane systems. Inspired by different biological mechanisms or by combining mathematical methods and/or ideas in computer science, a variety of P systems have been proposed in the past two decades [3–11]. These can be roughly classified as cell-like, tissue-like, and neural-like P systems. It has been proved that many P systems and variants are Turing complete (equivalent to a Turing machine) and effective (capable of solving NP-hard problems in a feasible time). Moreover, they have been applied to solve real-world problems [12,13], such as in machine learning [14–17], image- and signal-processing [18–24], robots [25,26], ecology, and system biology [27–29].

1.1. Related work

As one of the main forms of neural-like P systems, spiking neural P (SNP) systems, as proposed by Ionescu et al. [30], were abstracted from the biological fact that neurons handle and exchange spikes with each other along synapses. From the perspective of topological structure, an SNP system can be expressed as a directed graph, where the neurons are the nodes of the graph and the synapses correspond to the arcs between them. In addition to its topological structure, an SNP system contains two important components: data and firing rules. The data (denoted by a configuration vector) are used to describe the states of neurons, while the rules (spiking and/or forgetting rules) are used to characterize the dynamic behavior of the system. The firing rules have the form $E/a^c \rightarrow a^p$, where E denotes the regular expression. The semantics of the firing rule $E/a^c \rightarrow a^p$ can be explained as follows. Suppose that a neuron where the firing rule resides has n spikes. If $a^n \in L(E)$ and $n \geq c$, then the rule is enabled and the neuron fires. Note that $L(E)$ denotes the set of languages generated by the regular expression E. When the firing rule is applied, c spikes are removed from the neuron (hence $n - c$ spikes remain) and p new spikes are generated. Then the generated p spikes are sent to its consequent neurons. If $p = 0$, then the rule is known as a forgetting rule, written as $E/a^c \rightarrow \lambda$, where λ denotes the empty string. As described above,

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2019.105064.

☆☆ This work was partially supported by the National Natural Science Foundation of China (No. 61472328), Research Fund of Sichuan Science and Technology Project (No. 2018JY0083), Chunhui Project Foundation of the Education Department of China (Nos. Z2016143 and Z2016148), and Research Foundation of the Education Department of Sichuan province (No. 17TD0034), China.

* Corresponding author.
E-mail addresses: ph.xhu@hotmail.com (H. Peng), wj.xhu@hotmail.com (J. Wang).

https://doi.org/10.1016/j.knosys.2019.105064
0950-7051/

a firing rule $E/a^c \rightarrow a^p$ in a neuron has a firing condition: $a^n \in L(E)$. It is important to point out that the firing condition is only related to the state of the neuron, and does not depend on the states of other neurons. Neurons work in parallel, so SNP systems are distributed parallel computing models. Nondeterminism is an interesting characteristic of SNP systems: when two or more spiking rules in a neuron can be applied at the same time, one is nondeterministically chosen and applied. Moreover, SNP systems can work in three modes: generating, accepting, and computing.

A variety of SNP systems have been proposed. Motivated by the inhibitory and excitatory influence of astrocytes on synapses, Păun [31] and Pan et al. [32], discussed two SNP systems with astrocytes. Pan et al. [33] discussed SNP systems with anti-spikes by introducing anti-spikes that abstract from inhibitory impulses. Inspired by the biological fact that a synapse has one or more chemical channels, Peng et al. [34] presented SNP systems with multiple channels. Considering that the rules are placed in synapses instead of neurons, SNP systems with rules on synapses have been discussed by Song et al. [35], and another spike-consumption strategy was adopted by Peng et al. [36]. Chen et al. [37] investigated an axon P system where nodes were arranged in a linear structure and each only sent spikes to two neighbors. Inspired by the fact that every neuron has a positive or negative charge, SNP systems with polarizations have been discussed [38]. The structural dynamism of biological synapses inspired Cabarle et al. [39] to present an SNP system with scheduled synapses. Considering a new communication strategy among neurons, Pan et al. [40] discussed SNP systems with communication on request. With the limitation that at most one neuron works at each step, several sequential SNP systems were investigated by Ibarra et al. [41] and Zhang et al. [42]. Wang et al. [43] discussed an SNP system in which weights like those in artificial neural networks were introduced on synapses. Dynamic threshold neural P systems and coupled neural P systems were discussed by Peng et al. [44,45]. Moreover, using the thresholds instead of the original regular expression, Zeng et al. [46] proposed SNP systems with thresholds. Note that in SNP systems with thresholds, the firing condition is changed as $n \geq T$. A global clock is usually assumed in SNP systems, so they are synchronized. However, Cavaliere et al. [47] and Song et al. [48] investigated two asynchronous SNP systems. By integrating fuzzy logic in SNP systems, several fuzzy SNP systems have been developed, such as fuzzy reasoning SNP systems [49], weighted fuzzy SNP systems [50], and interval-valued fuzzy SNP systems [51].

Computational properties of variants of SNP systems have been investigated. As function-computing, natural number-generating, and language-generating devices, most have been proved to be Turing universal [52–55]. Furthermore, SNP systems have been applied to (theoretically) solve a number of computationally hard problems in a feasible (polynomial or linear) time. Application of SNP systems in some real-world problems, such as fault diagnosis [56–59], image processing [60], and combinatorial optimization [61], has recently received much attention.

1.2. Motivation

As mentioned above, data (i.e., states) and rules (firing and forgetting rules) are two important components in SNP systems and their variants. The firing condition $a^n \in L(E)$ is only related to the state of the neuron with which the firing rule is associated. Hence whether a neuron fires depends solely on its current state and has nothing to do with the states of other neurons. Therefore, the behavior of each neuron in an SNP system is controlled only by its state (the number of spikes). From this, an interesting idea emerges: can the firing of a neuron be controlled by the states of other neurons? Is there a biological fact that can support this interesting idea?

Two scientific findings in biological nervous systems [62,63] are related to this idea.

(1) Excitatory synapse [64]
The excitatory synapse transfers the excitation of presynaptic to postsynaptic. Once the action potential at the end of presynaptic fiber is reached, the excitatory synapse chemically or electrically passes it to the postsynaptic neurons and produces excitatory postsynaptic potentials. Excitatory postsynaptic potential is a depolarizing potential change that can be summed up by the activity of multiple excitatory synapses, and action potential is generated when it exceeds the threshold.
(2) Inhibitory synapse [65]
Presynaptic excitatory transmission has an inhibitory effect on postsynaptic excitation. When the excitatory (action) potential reaches the tip of the presynaptic fiber, it is transferred chemically or electrically to the postsynaptic neuron, where the inhibitory postsynaptic potential is produced. Inhibitory postsynaptic potential reduces the depolarization of the excitatory postsynaptic potential and thus inhibits action potential because of the short-circuit effect caused by hyperpolarization and increased ion permeability.

In SNP systems, spikes received by a neuron from other neurons are accumulated to update the state of the neuron, and if the firing condition is satisfied, then some new spikes will be generated. Therefore, the functioning of an excitatory synapse is basically consistent with the firing mechanism in SNP systems and their variants. However, the functioning of an inhibitory synapse is not reflected in these systems. The main motivation of this paper is to develop an inhibitory rule based on the functioning of inhibitory synapses and then to propose a new model of SNP systems, SNP systems with inhibitory rules (SNP-IRs).

An inhibitory rule has the form $(E_{en}, \overline{E_{in}})/a^c \rightarrow a^p$, where E_{en} is called enable regular expression, while E_{in} is called inhibitory regular expression. The semantics of inhibitory rules will be explained in detail later. The new firing condition can be written as $a^{n_1} \in L(E_{en}) \wedge a^{n_2} \notin L(E_{in})$, where n_1 is the number of spikes in the neuron where the inhibitory rule resides, and n_2 is the number of spikes in its preceding neuron (called an inhibitory neuron). Therefore, the firing condition indicates that the firing of an inhibitory rule in a neuron not only depends on its state but also is related to the state of other (inhibitor) neurons, which corresponds to the functioning of an inhibitory synapse.

The new variant differs from SNP systems in the following ways.

(1) The new variant introduces an inhibitory rule, inspired by the functioning of an inhibitory synapse.
(2) In the new variant, the firing of rules depends on both a neuron's state and those of other (inhibitor) neurons. However, the firing of rules in an SNP system is only related to a neuron's state. Therefore, the new variant seems to have more powerful control than an SNP system.
(3) Topologically, the new variant can be expressed as a directed graph with inhibitory arcs, which do not appear in standard SNP systems.

In summary, the novelty of the work is to abstract an inhibitory rule inspired by the functioning of an inhibitory synapse and to propose SNP systems with inhibitory rules.

The remainder of this paper is arranged as follows. Section 2 defines SNP-IR systems and provides an illustrative example. Section 3 studies the computational completeness of SNP-IR systems as number-generating/accepting and function-computing devices. Conclusions are drawn and further work is suggested in Section 4.

Please cite this article as: H. Peng, B. Li, J. Wang et al., Spiking neural P systems with inhibitory rules, Knowledge-Based Systems (2019) 105064, https://doi.org/10.1016/j.knosys.2019.105064.

2. SNP-IR systems

To define SNP-IR systems more clearly, some notions and notations related to both SNP systems and formal language theory are briefly reviewed. Further details can be found in [2,66].

Let Σ be an alphabet. The set of all of the finite strings over Σ is denoted by Σ^*, the empty string is written as λ, and the set of all of the nonempty strings over Σ is denoted by Σ^+. Usually, if $\Sigma = \{a\}$, then $\{a\}^*$ and $\{a\}^+$ can be simply written as a^* and a^+, respectively.

A regular expression over Σ is defined recursively: (i) λ and every $a \in \Sigma$ are regular expressions; (ii) if E_1, E_2 are two regular expressions over Σ, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over Σ; (iii) nothing else is a regular expression over Σ. A language $L(E)$ can be associated with regular expression E over Σ as follows: (i) $L(\lambda) = \lambda$ and $\forall\, a \in \Sigma$, $L(a) = \{a\}$; (ii) for any two regular expressions E_1, E_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 $L((E_1)^+) = (L(E_1))^+$.

2.1. Definition

Definition 1. An SNP-IR system of degree $m \geq 1$ is a tuple:

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

where:

(1) $O = \{a\}$ denotes a singleton alphabet (a is known as the spike);
(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are m neurons, denoted by $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
   (a) $n_i \geq 0$ denotes the number of spikes stored initially in neuron σ_i;
   (b) R_i denotes the finite set of rules of two types:
      (i) usual firing rules, of the form $E/a^c \rightarrow a^p$;
      (ii) inhibitory rules, of the form $(E_{en}, \overline{E_{in(i,j)}})/a^c \rightarrow a^p$,

   where $E, E_{en}, E_{in(i,j)}$ are the regular expressions over O, and $c \geq 1$, $p \geq 0$, and $c \geq p$;
(4) $syn = \{(i,j)\} \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i,i) \notin syn$ $\forall 1 \leq i \leq m$ (synapse connections);
(5) $in, out \in \{1, 2, \ldots, m\}$, respectively, distinguish input and output neurons.

From the perspective of a topological structure, the SNP-IR system is presented as a directed graph with inhibitory arcs, where m neurons are the nodes of the graph, and the synapses correspond to the arcs between the nodes. Specifically, if the system Π works in generative mode, then input neuron σ_{in} is removed from the system; by contrast, in accepting mode, output neuron σ_{out} is omitted.

As usual, the state of a neuron at time t is denoted by the number of spikes, while the state of the whole system at that time is characterized by the configuration vector $C_t = (n_1(t), n_2(t), \ldots, n_m(t))$, where $c_i(t)$ is the number of spikes in neuron σ_i, $1 \leq i \leq m$. Thus the initial configuration is $C_0 = (n_1, n_2, \ldots, n_m)$. As a result, a transition from one configuration to another can be defined. Usually, such a sequence of transitions starting from the initial configuration is known as a computation. If a configuration where no rule can be applied is attained in a computation, then it halts.

There are two types of rules in SNP-IR systems: usual firing rules and inhibitory rules. As in SNP systems, usual firing rules have the form $E/a^c \rightarrow a^p$. Suppose that a firing rule $E/a^c \rightarrow a^p$ is in neuron σ_i, and the neuron has $n_i(t)$ spikes. If $a^{n_i(t)} \in L(E)$ and $n_i(t) \geq c$, then the neuron fires and c spikes are removed from the neuron ($n_i(t) - c$ spikes are retained), and then p new spikes are generated. The generated spikes are sent to its succeeding neurons. If $q = 0$ in the rule, then it can written as $E/a^c \rightarrow \lambda$, which is known as a forgetting rule. If the forgetting rule is applied, then c spikes are removed but no spike is generated.

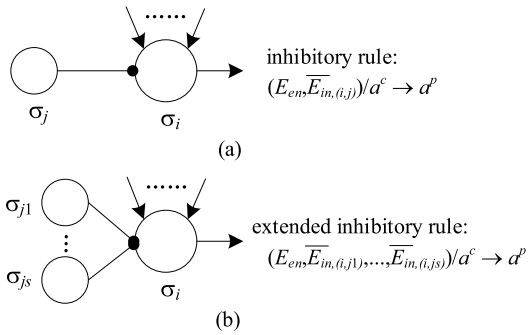

Fig. 1. Rules: (a) inhibitory rule, and (b) extended inhibitory rule.

Inhibitory rules are of the form $(E_{en}, \overline{E_{in,(i,j)}})/a^c \rightarrow a^p$, as shown in Fig. 1(a). Each inhibitory rule has two regular expressions to control the firing of a neuron: an enable regular expression E_{en} and an inhibitory regular expression $E_{in,(i,j)}$. Suppose that the inhibitory rule $(E_{en}, \overline{E_{in,(i,j)}})/a^c \rightarrow a^p$ is associated with neuron σ_i. The enable regular expression E_{en} is only related to neuron σ_i, and its firing condition is $a^{n_i(t)} \in L(E_{en})$. For the inhibitory regular expression $E_{in,(i,j)}$, its firing condition is $a^{n_j(t)} \notin L(E_{in,(i,j)})$, since it is only related to the state of neuron σ_j. Thus the firing condition of neuron σ_i can be denoted by a Boolean expression, $a^{n_i(t)} \in L(E_{en}) \wedge a^{n_j(t)} \notin L(E_{in,(i,j)})$. Note that in $E_{in,(i,j)}$, the subscript (i,j) indicates that there is an arc, called an inhibitory arc, between neurons σ_i and σ_j. An inhibitory arc that corresponds to an inhibitory synapse is denoted by an arc with a solid circle, as shown in Fig. 1(a), and neuron σ_j is therefore called the inhibitory neuron of σ_i. A directed arc with an arrow corresponds to an excitatory synapse. For neurons σ_i and σ_j, the following assumptions are given:

(1) If neurons σ_i and σ_j have an inhibitory arc, then there is no usual arc between them.
(2) If neuron σ_j is an inhibitory neuron of neuron σ_i, then neuron σ_j cannot send a spike to neuron σ_i. Moreover, the number of spikes in neuron σ_j cannot be changed even if some inhibitory rule in neuron σ_i is applied. Therefore, for neuron σ_i, neuron σ_j only plays the role of controlling the firing.

A neuron may have more than one inhibitory neuron; for example, neuron σ_i in Fig. 1(b) has s inhibitory neurons, $\sigma_{j1}, \sigma_{j1}, \ldots, \sigma_{js}$. The inhibitory rule of the form $(E_{en}, \overline{E_{in,(i,j1)}}, \ldots, \overline{E_{in,(i,js)}})/a^c \rightarrow a^p$ is called an extended inhibitory rule, whose firing condition can be denoted by $a^{n_i(t)} \in L(E_{en}) \wedge a^{n_{j1}(t)} \notin L(E_{in,(i,j1)}) \wedge \cdots \wedge a^{n_{js}(t)} \notin L(E_{in,(i,js)})$.

Based on the above firing mechanism, the state of neuron σ_i at time $t+1$ can be computed as

$$n_i(t+1) = \begin{cases} n_i(t) - c + n, & \text{if rule is used;} \\ n_i(t) + n, & \text{otherwise,} \end{cases} \quad (1)$$

where n is the number of spikes retrieved from other neurons. Note that whether a firing rule or inhibitory rule is applied, c spikes are removed from neuron σ_i.

In every computing step, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. The firing conditions of two rules in a neuron may be satisfied simultaneously, such as

Please cite this article as: H. Peng, B. Li, J. Wang et al., Spiking neural P systems with inhibitory rules, Knowledge-Based Systems (2019) 105064, https://doi.org/10.1016/j.knosys.2019.105064.

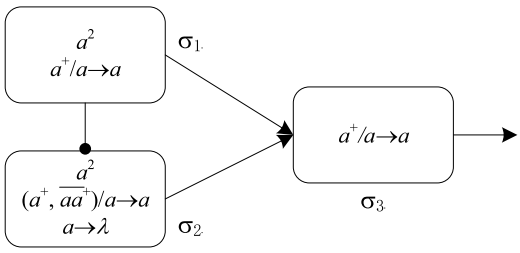

Fig. 2. An illustrative example.

two usual firing rules, two inhibitory rules, or a usual firing rule and an inhibitory rule. In either case, one of these enabled rules will be chosen nondeterministically. Consequently, the rules are applied sequentially in every neuron, whereas neurons work in parallel.

Any computing can correspond to a spike train consisting of zeros and ones, which describe the behavior of the output neuron: write 1 when the output neuron excites the spikes, and write 0 when the output neuron fires. Hence the number of steps between the first two spikes excited by the output neuron is regarded as the computational result. Let $N_2(\Pi)$ be the set of numbers computed by Π. $N_2SNP_m^n$ denotes the families of all sets $N_2(\Pi)$ computed by an SNP-IR system, consisting of at most m neurons and at most n rules in every neuron. Note that if one of m and n is not limited, then the symbol "*" is used to substitute it.

We can execute the SNP-IR system Π under the accepting mode: the spikes are received by an input neuron from the environment, and the output neuron is ignored. The system starts by importing the spike train from the environment, and then it stores the number n to a specified neuron in the form of $2n$ spikes. When the system halts, n is known as the number accepted by it. Denote by $N_{acc}(\Pi)$ the set of numbers accepted by the system Π, in which the subscript acc identifies that the system works in the accepting mode. $N_{acc}SNP_m^n$ denotes the family of all sets $N_{acc}(\Pi)$ accepted by an SNP-IR system consisting of at most m neurons and at most n rules in every neuron.

2.2. An illustrative example

An example that can generate a string of zeros and ones is provided to clarify the working mechanism of SNP-IR systems. Suppose an SNP-IR system has three neurons, σ_1, σ_2, and σ_3, as shown in Fig. 2. Note that $out = 3$, i.e., neuron σ_3 is an output neuron.

Initially, neurons σ_1 and σ_2 each have two spikes, while neuron σ_3 has no spike. Therefore, the initial configuration is $C_0 = (2, 2, 0)$.

At time 1, since neuron σ_1 has two spikes, rule $a^+/a \to a$ can be applied. After applying the rule, a spike will be consumed and a new spike will be generated and sent to neuron σ_3. Note that even if neuron σ_2 has two spikes that satisfy $a^2 \in L(a^+)$, two spikes in neuron σ_1 can inhibit the rule $(a^+, \overline{aa^+})/a \to a$ in neuron σ_2 because $a^2 \in L(aa^+)$, so no rule is enabled in neuron σ_2. Similarly, no rule is enabled in neuron σ_3. Therefore, $C_1 = (1, 2, 1)$.

At time 2, with a spike in neuron σ_3, rule $a^+/a \to a$ is applied to send a spike to the environment. With a spike in neuron σ_1, rule $a^+/a \to a$ is applied to send a spike to neuron σ_3. Since neuron σ_2 has two spikes and neuron σ_1 has one spike, the firing condition of rule $(a^+, \overline{aa^+})/a \to a$ is satisfied, i.e., $a^2 \in L(a^+) \wedge a^2 \notin L(aa^+)$, hence the rule is applied to send a spike to neuron σ_3. After neurons σ_1 and σ_2 fire, neuron σ_3 will receive two spikes. Thus $C_2 = (0, 1, 2)$.

At time 3, due to two spikes in neuron σ_3, rule $a^+/a \to a$ is applied to send a spike to the environment. Since neuron σ_2 has a spike and neuron σ_1 has no spike, rule $(a^+, \overline{aa^+})/a \to a$ and rule $a \to \lambda$ are enabled simultaneously. Therefore, one of the two rules is nondeterministically chosen and applied, and the following two cases are considered:

(1) If rule $(a^+, \overline{aa^+})/a \to a$ is applied, then neuron σ_2 sends a spike to neuron σ_3. Thus $C_3 = (0, 0, 2)$. At time 4, rule $a^+/a \to a$ in neuron σ_3 is applied to send a spike to the environment. Then neuron σ_3 again sends a spike to the environment. Therefore, the spike train generated by the system is "01111".
(2) If rule $a \to \lambda$ is applied, then the spike in neuron σ_2 is consumed by rule $a \to \lambda$. Thus $C_3 = (0, 0, 1)$. At time 4, rule $a^+/a \to a$ in neuron σ_3 is applied to send a spike to the environment. Therefore, the spike train generated by the system is "0111".

3. Computation completeness

In this section, we will discuss the universality of SNP-IR systems as devices for number generating, number accepting, and function computing. The universality of SNP-IR systems will be proven by the simulation of register machines, by which all recursively enumerable sets of numbers can be generated/accepted by SNP-IR systems. Moreover, a small universal function-computing device can be constructed.

A register machine is given as a tuple, $M = (m, H, l_0, l_h, I)$, where m indicates the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting instruction label, and I is the set of instructions. Each instruction in I is associated with a label in H. I has three types of instructions:

(1) $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then nondeterministically move to one of the instructions with labels l_j, l_k).
(2) $l_i : (SUB(r), l_j, l_k)$ (if register r is nonzero, then decrement it by 1 and move to the instruction with label l_j; otherwise, move to the instruction with label l_k).
(3) $l_h : HALT$ (halting instruction).

3.1. SNP-IR systems as number generating devices

A number n can be computed by a register machine working in the generating mode. Starting from the instruction with label l_0 and with all of the empty registers, the machine constantly applies instructions as distinguished by labels until it halts; at this moment, the number contained in the first register is known as the result computed by M. As we know, the family NRE can be characterized by register machines.

Theorem 1. $N_2SNP_*^2 = NRE$

Proof. Because $N_2SNP_*^2 \subseteq NRE$ is straightforward, it is only necessary to prove the inclusion $NRE \subseteq N_2SNP_*^2$. Hence a register machine $M = (m, H, l_0, l_h, I)$ working in the generative mode is considered. In general, all of the registers different from register 1 are assumed to be empty in the halting configuration, and during the computation, register 1 is never decremented.

To simulate register machine M, an SNP-IR system Π_1 is designed, which contains modules of three types: an ADD module to simulate the ADD instruction (Fig. 3), a SUB module to simulate the SUB instruction (Fig. 4), and a FIN module to output the computational result (Fig. 5).

Please cite this article as: H. Peng, B. Li, J. Wang et al., Spiking neural P systems with inhibitory rules, Knowledge-Based Systems (2019) 105064, https://doi.org/10.1016/j.knosys.2019.105064.

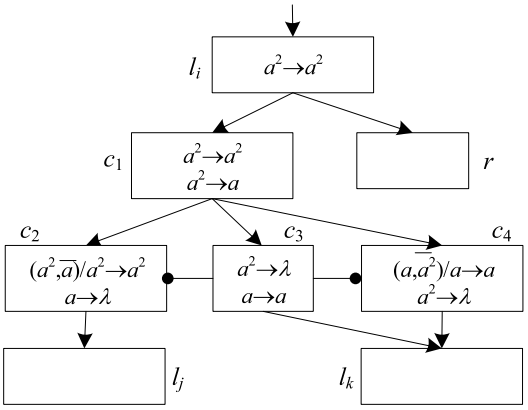

Fig. 3. ADD module, simulating the ADD instruction $l_i : (ADD(r), l_j, l_k)$.

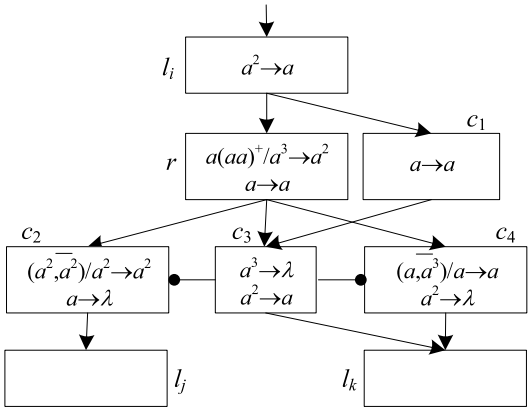

Fig. 4. SUB module, simulating the SUB instruction $l_i : (SUB(r), l_j, l_k)$.

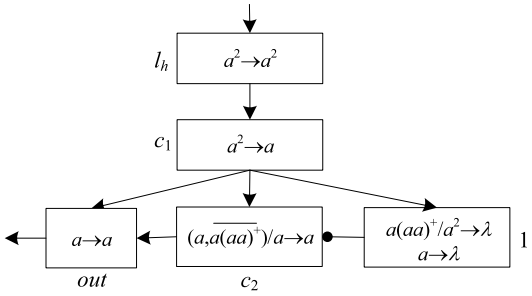

Fig. 5. FIN module.

Assume that every register r is associated with a neuron σ_r. We can code the number in register r: if register r stores the number $n \geq 0$, then neuron σ_r has $2n$ spikes. We associate a neuron σ_l with every instruction l in H, and we introduce some auxiliary neurons to these modules. Assume that each auxiliary neuron initially has no spike, and each neuron σ_{l_i} that is associated with l_i contains two spikes. Due to two spikes in neuron σ_{l_i}, the system Π_1 starts the simulation of the instruction $l_i : (OP(r), l_j, l_k)$ (OP identifies one of the operations ADD and SUB). Starting from the activation of neuron l_i, the simulation deals with neuron σ_r as identified by OP, and two spikes are introduced in one of the neurons σ_{l_j} and σ_{l_k}. The computation in M is continually simulated until neuron σ_{l_h} fires. During the computation, the spikes are sent into the environment twice, at times t_1 and t_2, and the computational result is the value $t_2 - t_1$ associated with the number contained in register 1.

To illustrate that the register machine M is correctly simulated by the system Π_1, we will discuss how ADD and SUB modules simulate ADD and SUB instructions, and how the computational result is exported by the FIN module.

(1) ADD module (Fig. 3) - simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$.

The system Π_1 starts from the simulation of instruction l_0, which is an ADD instruction. Assume that an ADD instruction $l_i : (ADD(r), l_j, l_k)$ is simulated at time t. At this moment, neuron σ_{l_i} has two spikes. Hence, rule $a^2 \to a^2$ is used to send two spikes to neurons σ_{c_1} and σ_r. Neuron σ_r receives two spikes, meaning that register r is incremented by 1. At time $t + 1$, with two spikes in neuron σ_{c_1}, rules $a^2 \to a^2$ and $a^2 \to a$ can be applied simultaneously. Therefore, one of the two rules in neuron σ_{c_1} is chosen nondeterministically. There exist the following two cases:

(i) At time $t+1$, if rule $a^2 \to a^2$ is used, then neuron σ_{c_1} sends two spikes to each of neurons σ_{c_2}, σ_{c_3}, and σ_{c_4}. At time $t + 2$, since neuron σ_{c_3} has two spikes, rule $(a^2, \overline{a}) \to a^2$ in neuron σ_{c_2} is enabled, but rule $(a, \overline{a^2}) \to a$ in neuron σ_{c_4} is inhibited. At time $t+3$, neuron σ_{l_j} receives two spikes from neuron σ_{c_2}, meaning that the system Π_1 starts to simulate instruction l_j. Simultaneously, two spikes in neurons σ_{c_3} and σ_{c_4} are consumed by rule $a^2 \to \lambda$.

(ii) At time $t+1$, if rule $a^2 \to a$ is used, then neuron σ_{c_1} sends a spike to neurons σ_{c_2}, σ_{c_3}, and σ_{c_4}. At time $t+2$, since neuron σ_{c_3} has a spike, rule $(a, \overline{a^2}) \to a$ in neuron σ_{c_4} is enabled, but rule $(a^2, \overline{a}) \to a^2$ in neuron σ_{c_2} is inhibited. At this time, rule $a \to a$ in neuron σ_{c_3} is enabled. At time $t + 3$, neuron σ_{l_k} receives two spikes in total from neurons σ_{c_3} and σ_{c_4}, meaning that the system Π_1 starts to simulate instruction l_k. Simultaneously, the spike in neuron σ_{c_2} is consumed by rule $a \to \lambda$.

Therefore, the ADD module can correctly simulate the ADD instruction: when neuron σ_{l_i} receives two spikes, the number of spikes in neuron σ_r is increased by two, and one of the neurons σ_{l_j} and σ_{l_k} is nondeterministically chosen.

(2) SUB module (Fig. 4) - simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$.

Suppose that a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is simulated at time t. At this moment, neuron σ_{l_i} has two spikes, hence its rule $a^2 \to a$ is used to send a spike to neurons σ_r and σ_{c_1}. At time $t + 1$, rule $a \to a$ in neuron σ_{c_1} is used to send a spike to neuron σ_{c_3}. Based on the number of spikes in neuron σ_r, there exist the following two cases:

(i) At time $t+1$, if neuron σ_r contains $2n+1$ (≥ 3) spikes (because the number in register r is n), then rule $a(aa)^+/a^3 \to a^2$ is used to send two spikes to neurons σ_{c_2}, σ_{c_3}, and σ_{c_4}. At time $t + 2$, since neuron σ_{c_3} has three spikes, rule $(a^2, \overline{a^2})/a^2 \to a^2$ in neuron σ_{c_2} is applied to send two spikes to neuron σ_{l_j}, but rule $(a, \overline{a^3})/a \to a$ in neuron σ_{c_4} is inhibited. Moreover, three spikes in neuron σ_{c_3} and two spikes in neuron σ_{c_4} are consumed. Hence neuron σ_{l_j} has two spikes, meaning that the system Π_1 starts to simulate instruction l_j.

(ii) At time $t+1$, if neuron c_r has only a spike (since the number in register r is still zero), rule $a \to a$ is applied and a spike is sent to each of neurons σ_{c_2}, σ_{c_3}, and σ_{c_4}. At time $t + 2$, since neuron σ_{c_3} has two spikes, rule $(a, \overline{a^3})/a \to a$ in neuron σ_{c_4} is applied to send a spike to neuron σ_{l_k}, but rule $(a^2, \overline{a^2})/a^2 \to a^2$ in neuron σ_{c_2} is inhibited. Moreover,

due to two spikes in neuron σ_{c_3}, $a^2 \to a$ is applied to send a spike to neuron σ_{l_k}, and a spike in neuron σ_{c_2} is consumed by rule $a \to \lambda$. Thus neuron σ_{l_k} has two spikes, meaning that the system Π_1 starts to simulate instruction l_j.

Consequently, the SUB instruction can be correctly simulated by the SUB module: the system starts from neuron σ_{l_i} receiving two spikes, and ends with the sending of two spikes to neuron σ_{l_j} (if the number in register r is greater than 0), or sending two spikes to neuron σ_{l_k} (if the number in register r is 0).

(3) FIN module (Fig. 5) — outputting the computation result.

Assume that neuron σ_{l_h} has two spikes at time t, meaning that M halts, and neuron σ_1 contains $2n$ spikes (i.e., register 1 contains the number n). Due to two spikes in neuron σ_{l_h}, rule $a^2 \to a^2$ is applied to send two spikes to neuron σ_{c_1}. At time $t+1$, neuron σ_{c_1} sends a spike to neurons σ_{c_2}, σ_{out}, and σ_1. At time $t+2$, due to a spike in neuron σ_{out}, rule $a \to a$ is applied to send the first spike into the environment. Note that neuron σ_1 is an inhibitory neuron of neuron σ_{c_2}. Since neuron σ_1 has $(2n+1) \geq 3$ spikes, rule $(a, \overline{a(aa)^+})/a \to a$ in neuron σ_{c_2} is inhibited. Moreover, two spikes in neuron σ_1 are consumed by rule $a(aa)^+/a^2 \to \lambda$. The process is repeated until neuron σ_1 has only a spike.

At time $t+n+1$, since neuron σ_1 has only a spike, rule $(a, \overline{a(aa)^+})/a \to a$ in neuron σ_{c_2} is applied to send a spike to neuron σ_{out}. At this time, the spike in neuron σ_1 is removed by rule $a \to \lambda$. At time $t+n+2$, neuron σ_{out} sends a spike to the environment. Consequently, the interval between the two spikes sent to the environment by the system is $(t+n+2)-(t+2) = n$, which indicates exactly the number in register 1 when M halts.

From the discussion above, the system Π_1 correctly simulates the register machine M working in generating mode, in which each neuron contains two rules at most. Therefore, the theorem holds. □

3.2. SNP-IR systems as number accepting devices

The number n can usually be accepted by a register machine working in the accepting mode as follows. The machine first reads a spike train that codes a number from the environment and stores it in the first register, and all of the registers are assumed to be empty. Then, starting from the instruction with label l_0, the machine continually uses the instructions as identified by labels. When the halting instruction is reached, the number is said to be accepted by M.

Theorem 2. $N_{acc}SNP^2_* = NRE$

Proof. An SNP-IR system Π_2 working in accepting mode is designed to simulate the deterministic register machine $M = (m, H, l_0, l_h, I)$. The proof will be described by a modification of the proof of Theorem 1. The system Π_2 contains modules of three types: a deterministic ADD module, a SUB module, and an INPUT module.

Fig. 6 shows the INPUT module. Neuron σ_{in} is used to read spike train $10^{n-1}1$ from the environment, where the interval between the two spikes in the spike train is $(n+1)-1 = n$, which is the number to be accepted.

Suppose that at time t, neuron σ_{in} reads the first spike from the environment. At time $t+1$, rule $a \to a$ in neuron σ_{in} is applied to send a spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. Note that neuron σ_{c_3} is the inhibitory neuron of neurons σ_{c_2} and σ_{c_1}. At time $t+2$, because neuron σ_{c_2} contains only a spike, rule $(a, \overline{a^2})/a \to a$ in neurons σ_{c_2} and σ_{c_1} is enabled. Neurons σ_{c_2} and σ_{c_1} send a spike to neuron σ_1, and they exchange a spike with each other. As a result, neuron σ_1 receives two spikes, and a spike is still retained in each of neurons σ_{c_2} and σ_{c_1}. The process is repeated until the second spike arrives at neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. During each step, neurons σ_{c_2} and σ_{c_1} exchange a spike with each other, and two spikes are added to neuron σ_1.

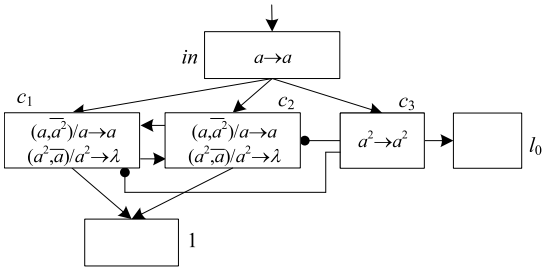

Fig. 6. INPUT module.

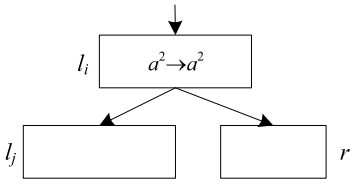

Fig. 7. ADD module, simulating $l_i : (ADD(r), l_j)$.

At time $t+n-1$, neuron σ_{in} reads the second spike from the environment. At time $t+n$, neuron σ_{in} sends a spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. At time $t+n+1$, since neurons σ_{c_1} and σ_{c_2} each have two spikes, the spikes are removed by rule $(a^2, \overline{a})/a^2 \to \lambda$. Moreover, neuron σ_{c_3} sends two spikes to neuron σ_{l_0}. Consequently, from time $t+2$ to time $t+n+1$, neuron σ_1 contains $2n$ spikes in total (i.e., the number of spikes in register 1 is n), and due to two spikes in neuron σ_{l_0}, the system starts to simulate the initial instruction, l_0.

In the case of accepting mode, deterministic ADD instructions, of the form $l_i : (ADD(r), l_j)$, are used in the register machine, as shown in Fig. 7. Note that there is no inhibitory neuron in this module. Suppose that two spikes are received by neuron σ_{l_i} at time t. At time $t+1$, due to two spikes in neuron σ_{l_i}, rule $a^2 \leftarrow a^2$ is applied to send two spikes to neurons σ_{l_j} and σ_r. Hence neuron σ_r has two spikes, meaning that register r is incremented by 1. With two spikes in neuron σ_{l_j}, the system starts to simulate instruction l_j.

Module SUB remains unchanged, as shown in Fig. 4. Module FIN is ignored, but neuron σ_{l_h} remains in the system. When neuron σ_{l_h} has two spikes, this indicates that halting instruction l_h is reached, and register machine M stops.

From the discussion above, the SNP-IR system correctly simulates the register machine working in accepting mode, where each neuron contains two rules at most. Therefore, the theorem holds. □

3.3. SNP-IR systems as function computing devices

A small universal SNP-IR system will be constructed to compute functions. The register machine $M = (m, H, l_0, l_h, I)$ used to compute the function $f : N^k \to N$ can be illustrated as follows. Initially, k arguments are introduced in k special registers (usually, the first k registers are used), and all of the registers are assumed to be empty. The machine starts from instruction l_0, and it executes constantly until the halting instruction l_h is reached. At this moment, the function value of f is the number stored in another special register, r_t. Denote by $(\varphi_0, \varphi_1, \ldots)$ a fixed admissible enumeration of the unary partial recursive functions.

l_0: $(SUB(1), l_1, l_2)$ l_1: $(ADD(7), l_0)$ l_2: $(ADD(6), l_3)$
l_3: $(SUB(5), l_2, l_4)$ l_4: $(SUB(6), l_5, l_3)$ l_5: $(ADD(5), l_6)$
l_6: $(SUB(7), l_7, l_8)$ l_7: $(ADD(1), l_4)$ l_8: $(SUB(6), l_9, l_0)$
l_9: $(ADD(6), l_{10})$ l_{10}: $(SUB(4), l_0, l_{11})$ l_{11}: $(SUB(5), l_{12}, l_{13})$
l_{12}: $(SUB(5), l_{14}, l_{15})$ l_{13}: $(SUB(2), l_{18}\}, l_{19})$ l_{14}: $(SUB(5), l_{16}, l_{17})$
l_{15}: $(SUB(3), l_{18}, l_{20})$ l_{16}: $(ADD(4), l_{11})$ l_{17}: $(ADD(2), l_{21})$
l_{18}: $(SUB(4), l_0, l_{22})$ l_{19}: $(SUB(0), l_0, l_{18})$ l_{20}: $(ADD(0), l_0)$
l_{21}: $(ADD(3), l_{18})$ l_{22}: $(SUB(0), l_{23}, l_{24})$ l_{23}: $(ADD(8), l_{22})$
l_{24}: $HALT$

Fig. 8. The small universal register machine M'_u.

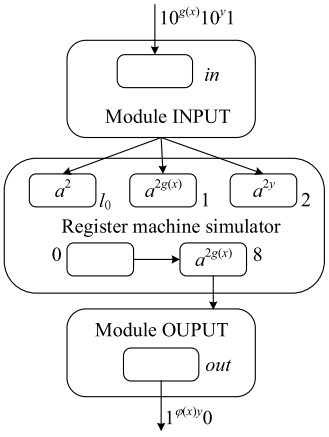

Fig. 9. General design of universal SNP-IR system Π_3.

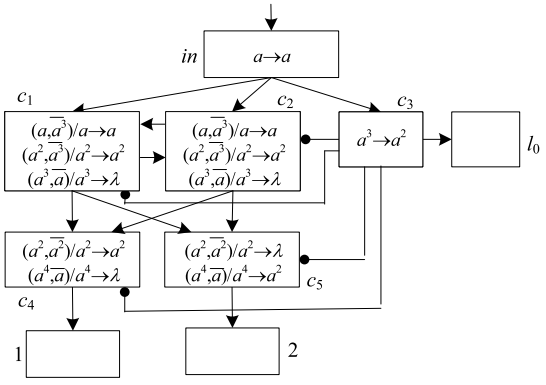

Fig. 10. INPUT module.

A register machine is said to be universal only if there exists a recursive function g such that $\varphi_x(y) = M_u(g(x), y)$ holds for all natural numbers x, y.

Korec [67] introduced a well-known small universal register machine for computing functions: $M_u = (8, H, l_0, l_h, I)$. The register machine M_u has 23 instructions and 8 registers (labeled 0 through 7). By importing two numbers $g(x)$ and y in registers 1 and 2, respectively, any $\varphi_x(y)$ can be computed by the register machine M_u; when the machine halts, the function value is stored in register 0. An SNP-IR system will be designed to simulate the register machine M_u. For simplicity, we modify the register machine M_u by adding a new register 8 and replacing the original halting instruction by l_{22} : $(SUB(0), l_{23}, l_h), l_{23}$: $(ADD(8), l_{22}), l_h$: $HALT$. Denote by M'_u the modification of M_u, as shown in Fig. 8. Therefore, register machine M'_u contains 24 ADD and SUB instructions, 9 registers, and 25 labels.

Theorem 3. *There exists a small universal SNP-IR system having 100 neurons for computing functions.*

Proof. We design an SNP-IR system Π_3 to simulate the universal register machine M'_u. The SNP-IR system Π_3 includes an INPUT module, an OUTPUT module, and several ADD and SUB modules to simulate the ADD and SUB instructions, respectively, of M'_u. The INPUT module is used to import a spike train from the environment, and the OUTPUT module exports the computational result.

Fig. 9 shows the general design of the universal SNP-IR system Π_3. Each register r in M'_u corresponds to a neuron σ_r, and if register r stores the number $n \geq 0$, then neuron σ_r has $2n$ spikes. Moreover, neuron σ_{l_i} in Π_3 corresponds to instruction l_i in M'_u. When neuron σ_{l_i} receives two spikes, it starts to simulate the instruction l_i. Once neuron σ_{l_h} receives two spikes, M'_u is completely simulated by the system Π_3. Finally, the first two spikes sent to the environment by the output neuron σ_{out} are regarded as the computational result (stored in register 8). In the initial configuration, assume that all of the neurons are empty.

Fig. 10 shows the INPUT module, which is used to read the spike train $10^{g(x)}10^y1$ from the environment, where $2g(x)$ spikes are stored in neuron σ_1 and $2y$ spikes are placed in neuron σ_2.

Assume that at time t_1, neuron σ_{in} receives the first spike from the environment. Similar to the analysis of the INPUT module in the proof of Theorem 2, at time $t_1 + 1$, rule $a \to a$ in neuron σ_{in} is applied to send a spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. Note that neuron σ_{c_3} is the inhibitory neuron of neurons σ_{c_2} and σ_{c_1}. At time t_2+2, because neuron σ_{c_3} contains only a spike, rule $(a, \overline{a^3})/a \to a$ is enabled in neurons σ_{c_1} and σ_{c_2} and applied to send a spike to neurons σ_{c_4} and σ_{c_5}, and they exchange a spike with each other. Note that neuron σ_{c_3} is also the inhibitory neuron of neurons σ_{c_4} and σ_{c_5}. At time $t_2 + 3$, with a spike in neuron σ_{c_3}, rule $(a^2, \overline{a^2})/a^2 \to a^2$ in neuron σ_{c_4} is enabled, but neuron σ_{c_5} is inhibited. Neuron σ_{c_4} sends two spikes to neuron σ_1, and two spikes in neuron σ_{c_5} are consumed by rule $(a^2, \overline{a^2})/a^2 \to \lambda$. The process is repeated until the second spike arrives at neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. During each step, two spikes in neuron σ_{c_4} are added to neuron σ_1. Therefore, from time $t_1 + 3$ to time $t_1 + g(x) + 2$, neuron σ_1 receives in total $2g(x)$ spikes (i.e., the number of spikes in register 1 is $g(x)$).

Assume that neuron σ_{in} receives the second spike at time t_2 (in fact, $t_2 = t_1 + g(x) + 2$). Similarly, at time $t_2 + 1$, rule $a \to a$ in neuron σ_{in} is applied to send a spike to neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. At this time, neurons σ_{c_1}, σ_{c_2}, and σ_{c_3} each have two spikes. At time $t_1 + 2$, since neuron σ_{c_3} has two spikes, rule

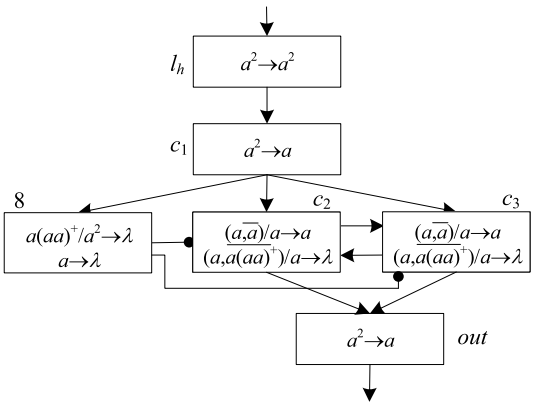

Fig. 11. OUTPUT module.

$(a^2, \overline{a^3})/a^2 \to a^2$ is enabled in neurons σ_{c_1} and σ_{c_2} and applied to send two spikes to neurons σ_{c_4} and σ_{c_5}, and they exchange two spikes with each other. At time t_1+3, due to two spikes in neuron σ_{c_3}, rule $(a^4, \overline{a})/a^4 \to a^2$ in neuron σ_{c_5} is enabled, but neuron σ_{c_4} is inhibited. Neuron σ_{c_5} sends two spikes to neuron σ_2, and four spikes in neuron σ_{c_5} are consumed by rule $(a^4, \overline{a^2})/a^4 \to \lambda$. The process is repeated until the third spike arrives at neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. During each step, two spikes in neuron σ_{c_5} are constantly added to neuron σ_2. Hence, from time t_2+3 to time t_2+y+2, neuron σ_2 receives in total $2y$ spikes (i.e., the number of spikes in register 2 is y). Since neuron σ_3 has three spikes, it fires to send two spikes to neuron σ_{l_0}, meaning that the system starts to simulate the initial instruction l_0.

From Fig. 8, we can observe that all of the ADD instructions have the form $l_i : (ADD(r), l_j)$. Consequently, a deterministic ADD module, whose working principle was discussed in the proof of Theorem 2, can be used in the simulation of the ADD instruction, as shown in Fig. 7.

The SUB module in Fig. 4 is used in the simulation of SUB instruction $l_i : (SUB(r), l_j, l_k)$. The working principle of the SUB module was described in the proof of Theorem 1.

Suppose that M halts now, i.e., the instruction l_h is reached. The computational result is contained in register 8, and it never decreases during the computation. The computational result is exported by an OUTPUT module, as shown in Fig. 11.

Assume that neuron σ_{l_h} has two spikes at time t, meaning that M halts, and neuron σ_8 contains $2n$ spikes (indicating the number n in register 8). At time $t+1$, neuron σ_{l_h} sends two spikes to neuron σ_{c_1}. At time $t+2$, neuron σ_{c_1} sends a spike to neurons σ_8, σ_{c_2}, and σ_{c_3}. Thus neuron σ_8 receives a spike, and it contains an odd number of spikes. Note that neuron σ_8 is the inhibitory neuron of neurons σ_{c_2} and σ_{c_3}. At time $t+3$, rule $(a, \overline{a})/a \leftarrow a$ in neurons σ_{c_2} and σ_{c_3} is applied to send a spike to neuron σ_{out}, and they exchange a spike with each other. At this time, two spikes are consumed in neuron σ_8 (i.e., the number in register 8 is decremented by 1). At time $t+4$, neuron σ_{out} sends the first spike to the environment. The process is repeated until only one spike is stored in neuron σ_8, and neuron σ_{out} sends a spike to the environment each time. At time $t+n+3$, because only one spike is in neuron σ_8, and neurons σ_{c_2} and σ_{c_2} each have a spike, rule $(a, \overline{a(aa)^+})/a \leftarrow \lambda$ is enabled. Thus the spike in neurons σ_{c_2} and σ_{c_2} is consumed. Consequently, from step $t+4$ to step $t+n+3$, neuron σ_{out} sends in total n spikes to the environment, which is exactly the number contained in register 8 when M halts.

From the discussion above, the system Π_3 correctly simulates the register machine M'_u. In SNP-IR system Π_3, we use a total of 100 neurons: (i) six neurons for the INPUT module; (ii) four neurons for the OUTPUT module; (iii) 56 auxiliary neurons for 14 SUB instructions; (iv) nine neurons for nine registers; and (v) 25 neurons for 25 instructions. □

Table 1
Comparison of different computing models in terms of small numbers of computing units.

Computing models	Number of neurons
SNP-IR systems	100
PSNP systems [38]	200
SNQ P systems with one type of spike [40]	181
Recurrent neural networks [68]	886

Theorem 3 shows a small number of computing units (i.e., neurons) for SNP-IR systems as function-computing devices to achieve Turing universality. To further evaluate the computational power of SNP-IR systems, Table 1 compares the proposed variant with other computing models in terms of small numbers of computing units. From Table 1, we can observe that recurrent neural networks [68], PSNP systems [38], and SNQ P systems with one type of spike [40] need 886, 200, and 181 neurons, respectively, to achieve Turing universality for the computing function, and SNP-IR systems need fewer neurons than all of these.

4. Conclusions and further work

We have proposed a new model of SNP systems, SNP-IR systems, which introduce inhibitory rules. These are inspired by the mechanism of inhibitory synapses, whereas usual firing rules are related to the mechanism of the excitatory synapse. Different from usual firing rules, the firing condition of an inhibitory rule not only depends on the state of the neuron where the rule resides, but also is related to the states of its inhibitory neurons. Therefore, whether a neuron fires in SNP-IR systems may be controlled by other neurons, i.e., its inhibitory neurons. This interesting feature, which SNP systems lack, indicates that SNP-IR systems have a stronger control ability. From the perspective of a directed graph, if neuron σ_j is an inhibitory neuron of neuron σ_i, then there exists a special arc from neuron σ_j to neuron σ_i, called an inhibitory arc. Different from usual arcs, inhibitory arcs do not transmit spikes; they only play the role of controlling a neuron's firing. The computational power of SNP-IR systems was investigated. The universality of SNP-IR systems as number-accepting/generating devices was proved. Moreover, we established a small universality result of SNP-IR systems for computing functions: a small universal function-computing device consisting of 100 neurons was constructed.

As stated in the existing SNP systems, some problems that refer to SNP-IR systems will be discussed, such as language generator, and asynchronous and sequential modes. Moreover, it is worth studying how to integrate other mechanisms or strategies in SNP-IR systems, so as to propose more models.

As stated above, since inhibitory rule of the form $(E_{en}, \bar{E_{in}})/a^c \to a^p$ is introduced SNP-IR systems, the firing of a neuron is controlled by both its own state and the states of its inhibitory neurons. However, the firing of a neuron in the existing SNP systems is controlled only by its own state. Therefore, SNP-IR systems can provide a stronger control ability than the existing SNP systems. This is a new and interesting attribute, which makes them more suitable for dealing with some practical application problems, for example, supervisory control problems in discrete event systems. The goal of supervisory control is to restrict the behavior of a system to satisfy the desired control specifications, such as deadlock avoidance or liveness enforcement in discrete event system. However, when the existing SNP systems are used

Please cite this article as: H. Peng, B. Li, J. Wang et al., Spiking neural P systems with inhibitory rules, Knowledge-Based Systems (2019) 105064, https://doi.org/10.1016/j.knosys.2019.105064.

to describe the discrete event system, they may suffer from the state explosion problem. Due to inhibitory rules, SNP-IR systems have a potential advantage to simplify the structure of model and reduce the corresponding state space so as to achieve the optimal supervisory control. Therefore, future work will address the application of SNP-IR systems in supervisory control problems in discrete event systems.

Acknowledgments

The authors thank the anonymous reviewers for providing very insightful and constructive suggestions, which have greatly help improve the presentation of this paper.

References

[1] G. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143, http://dx.doi.org/10.1006/jcss.1999.1693.
[2] G. Paun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, Inc., 2010.
[3] R. Freund, G. Păun, M. Pérez-Jiménez, Tissue-like P systems with channel-states, Theoret. Comput. Sci. 330 (1) (2005) 101–116, http://dx.doi.org/10.1016/j.tcs.2004.09.013.
[4] F. Bernardini, M. Gheorghe, Population P systems, J. UCS 10 (5) (2004) 509–539, http://dx.doi.org/10.3217/jucs-010-05-0509.
[5] G. Păun, R. Păun, Membrane computing and economics: numerical P systems, Fund. Inform. 73 (1,2) (2006) 213–227, http://dx.doi.org/10.3217/jucs-010-05-0509.
[6] L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, G. Vaszil, Variants of P colonies with very simple cell structure, Int. J. Comput. Commun. Control IV (3) (2009) 224–233, http://dx.doi.org/10.15837/ijccc.2009.3.2430.
[7] A. Spicher, O. Michel, M. Cieslak, J. Giavitto, P. Prusinkiewicz, Stochastic P systems and the simulation of biochemical processes with dynamic compartments, BioSystems 91 (3) (2008) 458–472, http://dx.doi.org/10.1016/j.biosystems.2006.12.009.
[8] M. Gheorghe, F. Ipate, R. Lefticaru, M.J. Pérez-Jiménez, A. Turcanu, L. Valencia-Cabrera, M. García-Quismondo, L. Mierlă, 3-col problem modelling using simple kernel P systems, Int. J. Comput. Math. 90 (4) (2013) 816–830, http://dx.doi.org/10.1080/00207160.2012.743712.
[9] G. Păun, M. Pérez-Jiménez, Solving problems in a distributed way in membrane computing: dP systems, Int. J. Comput. Commun. Control V (2) (2010) 238–250, http://dx.doi.org/10.1080/00207160.2012.743712.
[10] B. Song, C. Zhang, L. Pan, Tissue-like P systems with evolutional symport/antiport rules, Inform. Sci. 378 (C) (2016) 177–193, http://dx.doi.org/10.1016/j.ins.2016.10.046.
[11] Z. Zhang, T. Wu, A. Păun, L. Pan, Numerical P systems with migrating variables, Theoret. Comput. Sci. 641 (C) (2016) 85–108, http://dx.doi.org/10.1016/j.tcs.2016.06.004.
[12] G. Ciobanu, G. Păun, M. Pérez-Jiménez, Applications of Membrane Computing, Berlin: Springer-Verlag, 2006.
[13] G. Păun, M. Pérez-Jiménez, Membrane computing: brief introduction, recent results and applications, BioSystems 85 (1) (2006) 11–22, http://dx.doi.org/10.1016/j.biosystems.2006.02.001.
[14] Y. Zhao, X. Liu, J. Qu, The k-medoids clustering algorithm by a class of P system, J. Inf. Comput. Sci. 9 (18) (2012) 5777–5790.
[15] H. Peng, J. Wang, M. Pérez-Jiménez, A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, Inform. Sci. 304 (20) (2015) 80–91, http://dx.doi.org/10.1016/j.ins.2015.01.019.
[16] H. Peng, J. Wang, P. Shi, M. Pérez-Jiménez, A. Riscos-Núñez, An extended membrane system with active membrane to solve automatic fuzzy clustering problems, Int. J. Neural Syst. 26 (3) (2016) 1–17, http://dx.doi.org/10.1142/S0129065716500040.
[17] H. Peng, P. Shi, J. Wang, A. Riscos-Núñez, M. Pérez-Jiménez, Multiobjective fuzzy clustering approach based on tissue-like membrane systems, Knowl.-Based Syst. 125 (2017) 74–82, http://dx.doi.org/10.1016/j.knosys.2017.03.024.
[18] D. Díaz-Pernil, A. Berciano, F. Peña-Cantillana, M. Gutiérrez-Naranjo, Segmenting images with gradient-based edge detection using membrane computing, Pattern Recognit. Lett. 34 (8) (2013) 846–855, http://dx.doi.org/10.1016/j.knosys.2017.03.024.
[19] H. Peng, J. Wang, M. Pérez-Jiménez, Optimal multi-level thresholding with membrane computing, Digit. Signal Process. 37 (2015) 53–64, http://dx.doi.org/10.1016/j.dsp.2014.10.006.
[20] H. Peng, J. Wang, M. Pérez-Jiménez, P. Shi, A novel image thresholding method based on membrane computing and fuzzy entropy, J. Intell. Fuzzy Systems 24 (2) (2013) 229–237, http://dx.doi.org/10.3233/IFS-2012-0549.
[21] R.I. Yahya, S. Hasan, L. George, B. Alsalibi, Membrane computing for 2D image segmentation, Int. J. Adv. Soft Comput. Appl. 7 (1) (2015) 35–50.
[22] G. Zhang, M. Gheorghe, Y. Li, A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, Nat. Comput. 11 (4) (2012) 701–717, http://dx.doi.org/10.1007/s11047-012-9320-2.
[23] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, The framework of P systems applied to solve optimal watermarking problem, Signal Process. 101 (2014) 256–265, http://dx.doi.org/10.1016/j.sigpro.2014.02.020.
[24] J. Wang, P. Shi, H. Peng, Membrane computing model for IIR filter design, Inform. Sci. 329 (2016) 164–176, http://dx.doi.org/10.1016/j.ins.2015.09.011.
[25] C. Buiu, C. Vasile, O. Arsene, Development of membrane controllers for mobile robots, Inform. Sci. 187 (2012) 33–51, http://dx.doi.org/10.1016/j.ins.2011.10.007.
[26] X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate, R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, Integr. Comput.-Aided Eng. 23 (1) (2016) 15–30, http://dx.doi.org/10.3233/ICA-150503.
[27] M. Gheorghe, V. Manca, F. Romero-Campero, Deterministic and stochastic P systems for modelling cellular processes, Nat. Comput. 9 (2) (2010) 457–473.
[28] M. García-Quismondo, M. Levin, D. Lobo-Fernández, Modeling regenerative processes with membrane computing, Inform. Sci. 381 (2017) 229–249, http://dx.doi.org/10.1016/j.ins.2016.11.017.
[29] M. García-Quismondo, I. Nisbet, C. Mostello, M. Reed, Modeling population dynamics of roseate terns (sterna dougallii) in the northwest atlantic ocean, Ecol. Model. 68 (2018) 298–311, http://dx.doi.org/10.1016/j.ecolmodel.2017.12.007.
[30] M. Ionescu, G. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2–3) (2006) 279–308.
[31] G. Păun, Spiking neural P systems with astrocyte-like control, J. UCS 13 (11) (2007) 1707–1721, http://dx.doi.org/10.3217/jucs-013-11-1707.
[32] L. Pan, J. Wang, H. Hoogeboom, Spiking neural P systems with astrocytes, Neural Comput. 24 (3) (2012) 805–825, http://dx.doi.org/10.1162/NECO_a_00238.
[33] L. Pan, G. Păun, Spiking neural P systems with anti-spikes, Int. J. Comput. Commun. Control IV (3) (2009) 273–282, http://dx.doi.org/10.15837/ijccc.2009.3.2435.
[34] H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Luo, X. Huang, Spiking neural P systems with multiple channels, Neural Netw. 95 (2017) 66–71, http://dx.doi.org/10.1016/j.neunet.2017.08.003.
[35] T. Song, L. Pan, G. Păun, Spiking neural P systems with rules on synapses, Theoret. Comput. Sci. 529 (2014) 82–95, http://dx.doi.org/10.1016/j.tcs.2014.01.001.
[36] H. Peng, R. Chen, J. Wang, X. Song, T. Wang, F. Yang, Z. Sun, Competitive spiking neural P systems with rules on synapses, IEEE Trans. NanoBiosci. 16 (8) (2018) 888–895, http://dx.doi.org/10.1109/TNB.2017.2783890.
[37] H. Chen, T.-O. Ishdorj, G. Păun, Computing along the axon, Prog. Nat. Sci. 17 (4) (2007) 417–423.
[38] T. Wu, A. Păun, Z. Zhang, L. Pan, Spiking neural P systems with polarizations, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2018) 3349–3360, http://dx.doi.org/10.1109/TNNLS.2017.2726119.
[39] F. Cabarle, H. Adorna, M. Jiang, X. Zeng, Spiking neural P systems with scheduled synapses, IEEE Trans. Nanobiosci. 27 (5) (2016) 1337–1347, http://dx.doi.org/10.1109/TNB.2017.2762580.
[40] L. Pan, G. Păun, G. Zhang, F. Neri, Spiking neural P systems with communication on request, Int. J. Neural Syst. 28 (8) (2017) 1–13, http://dx.doi.org/10.1142/S0129065717500423.
[41] O. Ibarra, A. Păun, A. Rodríguez-Paón, Sequential SNP systems based on min/max spike number, Theoret. Comput. Sci. 410 (30) (2009) 2982–2991, http://dx.doi.org/10.1016/j.tcs.2009.03.004.
[42] X. Zhang, X. Zeng, B. Luo, L. Pan, On some classes of sequential spiking neural P systems, Neural Comput. 26 (5) (2014) 974–997, http://dx.doi.org/10.1162/NECO_a_00580.
[43] J. Wang, H. Hoogeboom, L. Pan, G. Păun, M. Pérez-Jiménez, Spiking neural P systems with weights, Neural Comput. 22 (2010) 2615–2646, http://dx.doi.org/10.1162/NECO_a_00022.
[44] H. Peng, J. Wang, M. Pérenz-Jiménez, A. Riscos-Núñez, Dynamic threshold neural P systems, Knowl.-Based Syst. 163 (2019) 875–884, http://dx.doi.org/10.1016/j.knosys.2018.10.016.
[45] H. Peng, J. Wang, Coupled neural P systems, IEEE Trans. Neural Netw. Learn. Syst. (2018) http://dx.doi.org/10.1109/TNNLS.2018.2871999.
[46] X. Zeng, X. Zhang, T. Song, L. Pan, Spiking neural P systems with thresholds, Neural Comput. 26 (7) (2014) 1340–1361, http://dx.doi.org/10.1162/NECO_a_00605.
[47] M. Cavaliere, O. Ibarra, G. Păun, O. Egecioglu, M. Ionescu, S. Woodworth, Asynchronous spiking neural P systems, Theoret. Comput. Sci. 410 (24) (2009) 2352–2364.

[48] T. Song, L. Pan, G. Păun, Asynchronous spiking neural P systems with local synchronization, Inform. Sci. 219 (2012) 197–207, http://dx.doi.org/10.1016/j.ins.2012.07.023.
[49] H. Peng, J. Wang, M. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy Reasoning spiking neural P system for fault diagnosis, Inform. Sci. 235 (2013) 106–116, http://dx.doi.org/10.1016/j.ins.2012.07.015.
[50] J. Wang, P. Shi, H. Peng, M. Pérez-Jiménez, T. Wang, Weighted fuzzy spiking neural P systems, IEEE Trans. Fuzzy Syst. 21 (2) (2013) 209–220, http://dx.doi.org/10.1109/TFUZZ.2012.2208974.
[51] J. Wang, H. Peng, W. Yu, J. Ming, M. Pérenz-Jiménez, C. Tao, X. Huang, Interval-valued fuzzy spiking neural P systems for fault diagnosis of power transmission networks, Eng. Appl. Artif. Intell. 82 (2019) 102–109, http://dx.doi.org/10.1016/j.engappai.2019.03.014.
[52] H. Chen, R. Freund, M. Ionescu, G. Păun, M. Pérez-Jiménez, On string languages generated by spiking neural P systems, Fund. Inform. 75 (1) (2007) 141–162.
[53] X. Zhang, X. Zeng, L. Pan, On string language generated by spiking neural P systems with exhaustive use of rules, Nat. Comput. 90 (1) (2008) 535–549, http://dx.doi.org/10.1007/s11047-008-9079-7.
[54] A. Păun, G. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 48–60, http://dx.doi.org/10.1016/j.biosystems.2006.06.006.
[55] A. Păun, M. Sidoroff, Sequentially induced by spike number in SNP systems: small universal machines, in: Membrane Computing, Springer, 2012, pp. 333–345.
[56] G. Xiong, D. Shi, L. Zhu, X. Duan, A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems, Math. Probl. Eng. 2013 (1) (2013) 211–244, http://dx.doi.org/10.1155/2013/815352.
[57] T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, M. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, IEEE Trans. Power Syst. 30 (3) (2015) 1182–1194, http://dx.doi.org/10.1109/TPWRS.2014.2347699.
[58] H. Peng, J. Wang, P. Shi, M. Pérez-Jiménez, A. Riscos-Núñez, Fault diagnosis of power systems using fuzzy tissue-like P systems, Integr. Comput.-Aided Eng. 24 (4) (2017) 401–411, http://dx.doi.org/10.3233/ICA-170552.
[59] H. Peng, J. Wang, J. Ming, P. Shi, M.J. Pérez-Jiménez, W. Yu, C. Tao, Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems, IEEE Trans. Smart Grid (2017) 1–8, http://dx.doi.org/10.1109/TSG.2017.2670602.
[60] D. Díaz-Pernil, F. Peña-Cantillana, M. Gutiérrez-Naranjo, A parallel algorithm for skeletonizing images by using spiking neural P systems, Neurocomputing 115 (2013) 81–91, http://dx.doi.org/10.1016/j.neucom.2012.12.032.
[61] G. Zhang, H. Rong, F. Neri, M. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, Int. J. Neural Syst. 24 (05) (2014) 1–16.
[62] J. Nicholls, A. Martin, P. Fuchs, D. Brown, M. Diamond, D. Weisblat, From Neuron to Brain, fifth ed., Sinuer Associtates, 2012.
[63] E. Kandel, J. Schwartz, T. Jessell, S. Siegelbaum, A. Hudspeth, Principles of Neural Science, fifth ed., McGraw-Hill, 2015.
[64] M. Sheng, C. Hoogenraad, Hoogenraad, The postsynaptic architecture of excitatory synapses: a more quantitative view, Annu. Rev. Biochem. 76 (1) (2007) 823–847, http://dx.doi.org/10.1146/annurev.biochem.76.060805.160029.
[65] C. Jean-Xavier, J. Pflieger, S. Liabeuf, L. Vinay, Inhibitory postsynaptic potentials in lumbar motoneurons remain depolarizing after neonatal spinal cord transection in the rat, J. Neurophysiol. 96 (5) (2006) 2274–2281, http://dx.doi.org/10.1152/jn.00328.2006.
[66] G. Rozenberg, A. Salomaa, Handbook of Formal Languages: Beyonds words, Springer Science & Business Media, 1997.
[67] I. Korec, Small universal register machines, Theoret. Comput. Sci. 168 (2) (1996) 267–301, http://dx.doi.org/10.1016/S0304-3975(96)00080-1.
[68] H. Siegelmann, E. Sontag, On the computational power of neural nets, J. Comput. System Sci. 50 (1) (1995) 132–150.

Please cite this article as: H. Peng, B. Li, J. Wang et al., Spiking neural P systems with inhibitory rules, Knowledge-Based Systems (2019) 105064, https://doi.org/10.1016/j.knosys.2019.105064.