Neural Process Lett
DOI 10.1007/s11063-014-9352-y

Homogenous Spiking Neural P Systems with Inhibitory Synapses

Tao Song · Xun Wang

Abstract Spiking neural P systems with inhibitory synapses (ISN P systems, for short) are a class of discrete neural-like computing models, which are inspired by the way of biological neurons storing and processing information and communication by means of excited and inhibitory impulses. In this work, we prove that ISN P systems can compute and accept any set of Turing computable numbers by using one type of neurons, thus can achieve Turing universality. Such systems are called homogenous ISN P systems. The results give a positive answer to an open problem left in (Pan and Păun, Int J Comput Commun 4(3):273–282, 2009) that "whether the number of types of neurons in universal SN P systems can be decreased by using inhibitory synapses". The obtained result is optimal in the sense of having minimal number of types of neurons in Turing universal SN P systems.

Keywords Membrane computing · P system · Spiking neural P system · Anti-spike · Turing completeness

1 Introduction

Natural Computing is known as a field of researching of human-designed computing inspired by nature [1]. *Membrane computing*, initialized in 1998 [2], is a new and hot branch of natural computing, whose aim is to construct powerful computing models and intelligent algorithms by abstracting ideas from a single living cell and from complexes of cells, such as tissues and organs including the human brain. The obtained computing systems are called P systems [2–4], and the algorithms are generally called membrane-inspired algorithm (or membrane algorithm) [5,6]. In recent years, membrane computing has developed rapidly (Information

T. Song
School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

X. Wang (✉)
Graduate School of Systems and Information Engineering, University of Tsukuba,
Tsukuba , Ibaraki 3050006, Japan
e-mail: wangxun0830@hotmail.com

Published online: 22 April 2014

Sciences Institute, ISI, considered membrane computing as "fast emerging research area in computer science" in 2003) [7]. Till now, there are three classes of P systems mainly investigated: cell-like P systems [2] and tissue-like P systems [3] and neural-like P systems [4]. Spiking neural P systems (SN P systems, for short) are a class of neural-like P systems, which are inspired by the way of biological neuron in human brain processing information and communicating with each other by means of electrical impulses (spikes) [4].

An SN P system consists of a set of *neurons* placed in the nodes of a directed graph, where neurons send signals (*spikes*) along *synapses* (arcs of the graph). A neuron may contain a number of *spikes* (a multiset over a singleton set) and *spiking/forgetting* rules (in form of productions in formal language theory). The information in a neuron is encoded in form of spikes. The neurons use spiking/forgetting rules to process the information inside. A neuron can contain more than one spiking/forgetting rules. Each rule is associated with a specific application context. In this way, a neuron can choose different rules to use when it contains different information, thus has information "self-classifying" function. This makes the neuron rather powerful working as the information processing unit.

The system works globally parallel, that is, different neurons can fire at the same moment, but the work in a neuron is sequential (at most one rule can be applied in every neuron in each time spiking). The *output neuron* can also send spikes to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system. We can define computation result of the system in various ways by the spike train.

In the research of neural networks, many results in the theoretical level have been obtained, such as Pollack demonstrated that second-order recurrent neural networks can work as dynamical recognizer for formal languages [10]; Giles proposed a approach by training recurrent neural networks with inserted rules to learn to recognize regular languages [11]; Siegelmann constructed a Turing universal neural nets [8]; Hyoetyniemi proposed that Turing machines can work as recurrent neural networks [12]; Moore describe methods of using fractal sets to keep track of arbitrary stack computations in neural units [13]. These theoretical results play an important role for implementations of neural systems.

In 1997, Maass proposed the third generation of neural network models – networks of spiking neurons [14]. It was proved that networks of spiking neurons performs well on doing computation, such as a network of spiking neurons with k adjustable delays is able to compute a much richer class of functions than a threshold circuit with k adjustable weights [15]; a network of spiking neurons cannot only perform all computations that a certain subclass of finite state machines can perform, but can also learn to do so [16]. Till now, a lots of works have been contributed to spiking neural networks in both theories and applications (one can refer to [17] for details.

SN P systems, as a new candidate of spiking neural networks, perform well on doing computation. For instance, Siegelmann constructed a Turing universal (do what Turing machine can do) neural networks with 886 neurons in [8], while for SN P systems 10 neurons are sufficient to achieve the Turing universality [9]. From the computation point of view, the computability of SN P systems is worth to be deeply investigated. In the past years, many results have been obtained. The systems can be used as computing devices mainly in three ways: generating (computing) sets of numbers [18,19], generating languages [20,21] and computing recursive functions [22,23]. Moreover, SN P systems with neuron budding and cell division can (theoretically) generate exponential working space in linear time, thus can provide a way to solve computationally hard problems in a feasible (polynomial or linear) time (see, e.g., [24,25]). Some other applications of SN P systems were also developed [26]. Inspired by different biological facts, many variants of SN P systems have also been investigated, such

as asynchronous SN P systems [27], asynchronous SN P systems with local synchronization [28], SN P systems with astrocyte-like control [29–31], SN P systems with rules on synapses [32]. Inspired by the functioning of neurons processing information by inhibitory impulses among biological neurons, SN P systems with inhibitory synapses (ISN P systems for short) were proposed in [33]. In ISN P systems, a neuron can contain a number of spikes or anti-spikes (representing the information in the neuron), spiking and forgetting rules (which are used to process the information and send the information to the neighboring neurons).

In [33], several open problems on ISN P systems are left. Some answers of the problems have been investigated in [34]. Specifically, it was obtained that SN P system with anti-spikes are universal by using two categories of pure form of spiking rules and without using forgetting rules. The one that whether we can reduce the type of neurons in universal SN P systems by using inhibitory synapses is still open. The computation power of SN P systems with one type of neurons is important in the computability research of SN P systems. As well, universal SN P systems with one type of neurons can provide powerful neural computing models with uniform computing units.

In this work, we consider a restricted variant of ISN P systems, called homogenous ISN P systems (shortly called HISN P systems). In the systems, each neuron contains the same set of spiking and forgetting rules in the sense of having one type of neurons. By simulating the register machine (which is equivalent with Turing machine), we obtain that HISN P systems can achieve Turing universality as number generating and accepting devices, i.e., they can compute and accept the family of sets of numbers that can be computed and accepted by Turing machine.

This result gives a positive answer to the problem left—we can decrease the number of types of neurons to one in universal SN P systems by using inhibitory synapses. The result is optimal in the sense of having the minimal number of types of neurons in Turing universal SN P systems. The universal results have a good interpretation: the inhibitory function of synapses is crucial for the functioning of the system. SN P systems with one type of rules can achieve Turing universality by using inhibitory synapses.

2 Homogenous SN P Systems with Inhibitory Synapses

Inspired by the functioning of inhibitory impulses among biological neurons, SN P systems with anti-spikes and inhibitory synapses are introduced [33], where besides usual positive spikes denoted by a, anti-spikes denoted by $\bar{a}$ are also considered. Before we introduce homogenous SN P systems with inhibitory synapses (HISN P systems, for short), we recall some prerequisites. It is useful for readers to have some familiarity with basic elements of formal language theory, e.g., from [35].

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

A regular expression over an alphabet Σ is defined as follows: (i) λ and each $a \in \Sigma$ is a regular expression, (ii) if E_1, E_2 are regular expressions over Σ, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over Σ, and (iii) nothing else is a regular expression over Σ. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in \Sigma$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over Σ. Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can also be written as E^*.

We define now the HISN P system. The definition is complete, but familiarity with the basic elements of SN P systems with anti-spikes and inhibitory synapses (e.g. from [4,33,34]) is helpful.

An *HISN P system* of degree $m \geq 1$ is a construct of the form:

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

where

- $O = \{a, \bar{a}\}$ is the alphabet, where a is *spike* and $\bar{a}$ is *anti-spike*;
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons* of the form $\sigma_i = (n_i, R)$ with $1 \leq i \leq m$, where
  1. n_i is a natural number representing the *initial number of spikes* in neuron σ_i, which corresponds to the impulse level of neuron σ_i at the beginning of the computation;
  2. R is set of *rules* in each neuron of the following two forms:
     (a) $E/b^c \rightarrow b'$ is the spiking rule, where E is the regular expression over $\{a\}$ or $\{\bar{a}\}$, $b, b' \in \{a, \bar{a}\}$ and $c \geq 1$;
     (b) $b^s \rightarrow \lambda$ is the forgetting rule, with the restriction that for any $s \geq 1$ and any spiking rule $E/b^c \rightarrow b'$, $b^s \notin L(E)$ with $b \in \{a, \bar{a}\}$, where $L(E)$ is set of regular languages associated with regular expression E and λ is the empty string;
- $syn = syn_e \cup syn_i \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ is the set of *synapses* between neurons, where syn_e is the set of excited synapses and syn_i is the set of inhibitory synapses;
- $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, where the input neuron can receive spikes from the environment, and the output neuron can emit spikes out to the environment.

In HISN P systems, any neuron has the same set of rules in sense of having the homogeneity. A neuron may contain a number of spikes or anti-spikes, but not both of them. When spikes and anti-spikes meet in a neuron they will immediately annihilate each other in a maximal manner, Formally, if a neuron contains a^r spikes and $\bar{a}^s$ anti-spikes, then the annihilating rule $a\bar{a} \rightarrow \lambda$ can be immediately applied in a maximal manner, ending with $r - s$ spikes (if $r > s$) or $s - r$ anti-spikes (if $s > r$) in the neuron. This process happens immediately, which means at any time there are only spikes or anti-spikes in any neuron.

The rules of the form $r_{sp} = E/b^c \rightarrow b'$ are *spiking rules*, which can be applied in any neuron as follows: if neuron σ_i contains k spikes or anti-spikes b with $b^k \in L(E)$ and $k \geq c$, the rule r_{sp} is enabled to be applied. By using the rule, c spikes/anti-spikes b are consumed, thus $k - c$ spikes/anti-spikes b remain in the neuron, and one spike or anti-spike b' is sent to all neurons σ_j such that $(i, j) \in syn$. There are four categories of spiking rules identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$, where (b, b') indicates the spiking rules of the form $E/b^c \rightarrow b'$. For any spiking rule r_{sp}, if $L(E) = b^c$, the rule is simply written as $b^c \rightarrow b'$.

Every neuron can contain several spiking rules. Because two spiking rules, $E_1/b^{c_1} \rightarrow b'$ and $E_2/b^{c_2} \rightarrow b'$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more spiking rules are enabled in a neuron at some moment, while only one of them is chosen non-deterministically to use. It is important to notice that the applicability of a spiking rule is controlled by checking the number of spikes or anti-spikes contained in the neuron against a regular expression over $\{a\}$ or $\{\bar{a}\}$ associated with the rule.

Rules of the form $b^s \rightarrow \lambda, s \geq 1$ are *forgetting rules* with the restriction $b^s \notin L(E)$, where $L(E)$ is set of regular languages associated with regular expression E and λ is the empty string. It means if a spiking rule is applicable, then no forgetting rule is applicable,

and vice versa. If neuron σ_i contains exactly s spikes/anti-spikes, the forgetting rule $b^s \to \lambda$ can be applied, by which s spikes or anti-spikes can be removed from the neuron (becoming empty string).

A global clock is assumed, marking the time for all neurons. In each time unit, if a neuron σ_i can use one of its rules, then a rule from R must be used. Thus, the rules are used in a sequential manner in each neuron, but neurons function in parallel with each other.

The set of synapses is denoted by $syn = syn_e \cup syn_i \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$, where syn_e is the set of excited synapses and syn_i is the set of inhibitory synapses. Specifically, if neuron σ_i fires at certain step sending a spike to neuron σ_j along the excited synapse (i, j), neuron σ_j will receive a spike; if inhibitory synapse (i, j) is a inhibitory synapse, neuron σ_j will receive an anti-spike.

The configuration of HISN P systems is of the form $\langle c_1, c_2, \ldots, c_m \rangle$ with $c_i \in \mathbb{Z}$. If $c_i \geq 0$, it means that there are c_i spikes in neuron σ_i; if $c_i < 0$, it indicates there are anti-spikes in neuron σ_i. The initial configuration of the system is $\langle n_1, n_2, \ldots, n_m \rangle$. By using the rules above, one can define *transitions* among configurations. A series of transitions starting from the initial configuration, halting or not, is called a *computation*. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system. With any computation, halt or not, we associate a spike train. It is assumed that output neuron sends out only spikes, not also anti-spikes. This restriction is natural/elegant, but not essential.

The *result of a computation* of the system Π is defined as the time internal of first two spikes being emitted to the environment by the output neuron at steps t_1, t_2, in the form of number $t_2 - t_1$; we say that this number is computed or generated by Π. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript 2 indicates that we only consider the distance between the first two spikes of any computation.

An HISN P system Π can also work in the accepting mode. A number n is introduced in a specified neuron in the form of $f(n)$ spikes by reading spike train from the environment through the input neuron. 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 $N_{acc}(\Pi)$ (the subscript acc indicates the system works in the accepting mode.

We denote by $N_\alpha HISNP_m(cate_l, rule_k, forg_h, Inh), \alpha \in \{2, acc\}$ all sets of numbers generated or accepted by HISN P systems of degree m, where at most l categories of spiking rules are used, each neuron contains at most k rules, forgetting rules removes at most h spikes each time and inhibitory synapses can be used to connect neurons. If the forgetting rules are not used, the indication of $forg_h$ will be removed from the notation. If the parameter m is not bounded, then it is replaced with $*$.

3 Universality Results

In this section, we prove that HISN P systems with no forgetting rule and only one category of spiking rules can generate (compute) and accept any set of Turing computable natural numbers, thus achieve computational completeness of HISN P systems.

In the following university proofs, *register machine* is applied. Formally, a register machine is a 5-tuple $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is a set of labels of the instructions, l_0 is the label of initial instruction, l_h is the label of halting instruction, and R is the set of instructions (each label from H labels a unique instruction from R). The instructions are of the following forms:

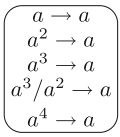

Fig. 1 The neuron in Π

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

The register machine computes number n as follows: starting M with all registers empty, the initial instruction with label l_0 is applied, and the instructions are applied as indicated by labels. When the register machine M proceeds to the halt instruction, number n stored at that time in the first register is said to be computed by M. In acceptive mode, a random number is stored in the first register with other registers being empty. If the computation starting from the initial configuration eventually halts, the number is said to be accepted by M.

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.) Register machine can characterize the family of Turing computable sets of numbers NRE in both generative and acceptive mode [36], where in the acceptive mode, it is sufficient to accept NRE by using deterministic instruction l_i : (ADD(r), l_j).

Without loss of generality, it can be assumed that l_0 labels an ADD instruction, that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation (its content is only added to). When we compare the power of two number generating/accpeting devices, we use the convention that number zero is ignored, which corresponds to the usual practice of ignoring the empty string when comparing the power of two devices in language and automata theory.

In the following proofs, the HISN P systems are represented graphically, where a rounded rectangle with initial number of spikes inside are used to represent a neuron without specifying the set of rules in each neuron, since any neuron has the same set of rules. Each inhibitory synapse in the systems is marked by a "black dot".

Theorem 3.1 $N_2HISNP(cate_1, rule_5, Inh) = NRE$ *with* $cate_1 = \{(a, a)\}$.

Proof It is enough to prove the inclusion $NRE \subseteq N_2HASNP$, for the converse inclusion is straightforward. (we can prove it by the similar technical details in Sect. 8.1 in [37], but is cumbersome).

Let M be a register machine. An HISN P system Π is constructed to simulate M, where the homogenous set of spiking rules is $R = \{a \to a, a^2 \to a, a^3 \to a, a^3/a^2 \to a, a^4 \to a\}$, and the neuron in Π is shown in Fig. 1.

In HISN P system Π, each register r of M is associated with a neuron σ_r, and if the number stored in register r is $n \geq 0$, neuron σ_r will contain $n + 5$ spikes. Moreover, each instruction l_i of M is represented by an neuron σ_{l_i}, and if machine M reaches the instruction l_i during the computation, neuron σ_{l_i} will be activated to start to simulate the operation of l_i. When register machine M reaches the halting instruction, neuron σ_{l_h} is activated, which means the FIN module begin to output the computing result of the system.

Module ADD shown in Fig. 2—simulating the ADD instruction l_i : (ADD(r), l_j, l_k).

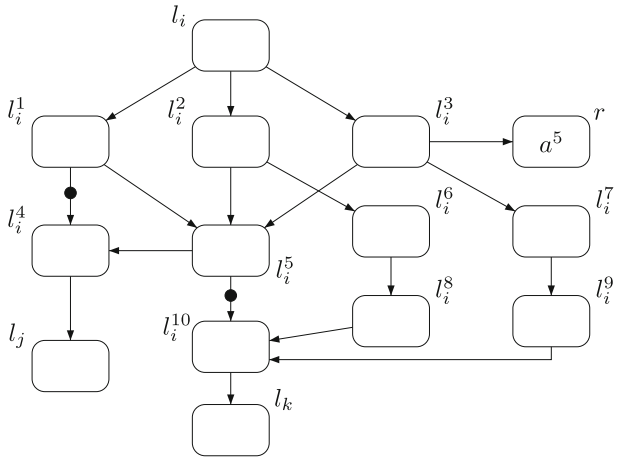

Fig. 2 Module ADD of Π (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

The initial instruction l_0 of M is an ADD instruction. Let us assume that at step t, an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ has to be simulated, with one spike in neuron σ_{l_i} and no spike in any other neurons, except in those neurons associated with registers. Having one spike inside, neuron σ_{l_i} fires at step t sending one spike to neurons $\sigma_{l_i^1}$, $\sigma_{l_i^2}$ and $\sigma_{l_i^3}$, respectively. In the next step, neuron $\sigma_{l_i^3}$ sends one spike to neuron σ_r, which simulates increasing the number in register r by one. At the same moment, neuron $\sigma_{l_i^5}$ receives three spikes, neurons $\sigma_{l_i^6}$ and $\sigma_{l_i^7}$ receive one spike, as well as neuron $\sigma_{l_i^4}$ receives an anti-spike (since the spike from neuron $\sigma_{l_i^1}$ passes along inhibitory synapse $(\sigma_{l_i^1}, \sigma_{l_i^4})$, it is changed to an anti-spike when it arrives in neuron $\sigma_{l_i^4}$). With three spikes inside, neuron σ_{c_5} non-deterministically chooses one of the two spiking rules $a^3 \to a$ and $a^3/a^2 \to a$ to apply at step $t+3$, which will non-deterministically activate neuron σ_{l_j} or σ_{l_k}.

- In neuron $\sigma_{l_i^5}$, if rule $a^3 \to a$ is chosen to be used, then it sends one spike to neurons $\sigma_{l_i^4}$ and $\sigma_{l_i^{10}}$, but when the spike arrives at neuron $\sigma_{l_i^{10}}$, it will be changed into an anti-spike due to the inhibitory synapse between neurons $\sigma_{l_i^5}$ and $\sigma_{l_i^{10}}$. In neuron $\sigma_{l_i^4}$, the anti-spike inside will be annihilated by the spike from neuron $\sigma_{l_i^5}$, so it keeps inactive in this case. At that time, with an anti-spikes inside, neuron $\sigma_{l_i^{10}}$ can not fire. At step $t+3$, two spikes are emitted to neuron $\sigma_{l_i^{10}}$ from neurons $\sigma_{l_i^8}$ and $\sigma_{l_i^9}$. After the annihilation, neuron $\sigma_{l_i^{10}}$ has one spike and the rule $a \to a$ is enabled. Neuron $\sigma_{l_i^{10}}$ fires at step $t+4$, sending one spike to neuron σ_{l_k}, which can be activated in the next step to start to simulate instruction l_k of M.
- In neuron $\sigma_{l_i^5}$, if rule $a^3/a^2 \to a$ is chosen to use, then after the rule is applied, neuron $\sigma_{l_i^5}$ can fire for the second times by using rule $a \to a$. It indicates that neuron $\sigma_{l_i^4}$ receives two spikes and $\sigma_{l_i^{10}}$ receives anti-spikes in $t+2$ and $t+3$ two steps. The two anti-spikes in neuron $\sigma_{l_i^{10}}$ can be annihilated by the two spikes from neuron $\sigma_{l_i^8}$ and $\sigma_{l_i^9}$ arriving at step $t+3$. With no spike or anti-spike inside, neuron $\sigma_{l_i^{10}}$ keeps inactivated. In neuron $\sigma_{l_i^4}$, after annihilating the anti-spike inside, it has one spike and fires at step $t+4$, sending one spike to neuron σ_{l_j}. Neuron σ_{l_j} fires at step $t+5$ to start to simulate instruction l_j of M.

Table 1 The numbers of spikes in neurons of ADD module during the ADD instruction simulation with neuron σ_{l_j} finally activated

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	$n+5$	$n+5$	$n+6$	$n+6$	$n+6$	$n+6$
$\sigma_{l_i^1}$	0	1	0	0	0	0
$\sigma_{l_i^2}$	0	1	0	0	0	0
$\sigma_{l_i^3}$	0	1	0	0	0	0
$\sigma_{l_i^4}$	0	0	-1	0	1	1
$\sigma_{l_i^5}$	0	0	3	1	0	0
$\sigma_{l_i^6}$	0	0	1	0	0	0
$\sigma_{l_i^7}$	0	0	1	0	0	0
$\sigma_{l_i^8}$	0	0	0	1	0	0
$\sigma_{l_i^9}$	0	0	0	1	0	0
$\sigma_{l_i^{10}}$	0	0	0	-1	0	0
σ_{l_j}	0	0	0	0	0	1
σ_{l_k}	0	0	0	0	0	0

Therefore, from firing neuron σ_{l_i}, the system adds one spike to neuron σ_r and nondeterministically fires one of neurons σ_{l_j} and σ_{l_k}, which correctly simulates the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

The evolution of the numbers of spikes (positive numbers) or anti-spikes (negative numbers) in neurons of ADD module during the ADD instruction simulation is shown in Tables 1 and 2, where Tables 1 and 2 correspond to the two cases in which neuron σ_{l_j} or σ_{l_k} is activated, respectively.

Module SUB shown in Fig. 3—simulating the SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$.

Assume at step t, the system starts to simulate a SUB instruction l_i of M. Having one spike inside, neuron σ_{l_i} fires at step t emitting a spike to neurons $\sigma_{l_i^1}$, $\sigma_{l_i^2}$ and σ_r. The spike arriving at neuron σ_r will be changed into an anti-spike due to the inhibitory synapse (σ_{l_i}, σ_r). Similarly, neuron $\sigma_{l_i^2}$ receives an anti-spike and keeps inactive. One of the spikes in neuron σ_r can be annihilated by the anti-spike, and there are the following two cases in neuron σ_r.

- Before receiving the anti-spike, if the number stored in register r is not zero, then neuron σ_r contains at least 6 spikes. After the annihilation, the number of spikes in neuron σ_r is decremented by one, which simulates decreasing the number in register r by one. With more than 5 spikes inside, no rule is enabled in neuron σ_r, so it keeps inactive. Moreover, one spike arrives at neuron $\sigma_{l_i^{13}}$ at step $t+3$ by passing along path $\sigma_{l_i^1}, \sigma_{l_i^{11}}, \sigma_{l_i^{12}}$. One step later, neuron $\sigma_{l_i^{13}}$ fires sending one spike to neurons $\sigma_{l_i^2}$ and σ_{l_j}. The anti-spike in neuron $\sigma_{l_i^2}$ will be annihilated by the spike from neuron $\sigma_{l_i^{13}}$. By receiving the spike, neuron σ_{l_j} fires at step $t+5$ to start to simulate instruction l_j of M.
- Before receiving the anti-spike, if the number in register r is zero, then neuron σ_r has 5 spikes. After annihilating the anti-spike, neuron σ_r has 4 spikes and the rule $a^4 \to a$ is enabled. At step $t+1$, neuron σ_r fires sending one spike to neurons $\sigma_{l_i^2}$ and $\sigma_{l_i^3}$. In neuron $\sigma_{l_i^2}$, the anti-spike is annihilated, and it keeps inactive. At step $t+2$, neurons $\sigma_{l_i^3}$ sends a spike to neuron $\sigma_{l_i^4}$, which fires one step later emitting one spike to neurons $\sigma_{l_i^5}$ and $\sigma_{l_i^{13}}$,

Table 2 The numbers of spikes in neurons of ADD module during the ADD instruction simulation with neuron σ_{l_k} finally activated

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	$n+5$	$n+5$	$n+6$	$n+6$	$n+6$	$n+6$
$\sigma_{l_i^1}$	0	1	0	0	0	0
$\sigma_{l_i^2}$	0	1	0	0	0	0
$\sigma_{l_i^3}$	0	1	0	0	0	0
$\sigma_{l_i^4}$	0	0	-1	0	0	0
$\sigma_{l_i^5}$	0	0	3	0	0	0
$\sigma_{l_i^6}$	0	0	1	0	0	0
$\sigma_{l_i^7}$	0	0	1	0	0	0
$\sigma_{l_i^8}$	0	0	0	1	0	0
$\sigma_{l_i^9}$	0	0	0	1	0	0
$\sigma_{l_i^{10}}$	0	0	0	-1	1	0
σ_{l_j}	0	0	0	0	0	0
σ_{l_k}	0	0	0	0	0	1

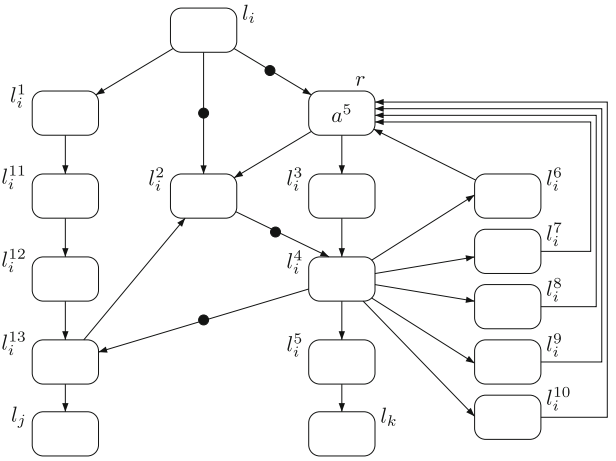

Fig. 3 Module SUB of Π (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$)

respectively. The spike passing to neuron $\sigma_{l_i^{13}}$ will be changed into an anti-spike by the inhibitory synapse, such that neuron $\sigma_{l_i^{13}}$ can not fire. At step $t+4$, neuron σ_{l_j} receives a spike from neuron $\sigma_{l_i^5}$ and becomes active in the next step, which starts to simulate instruction l_k of M. It is worth to point out that 5 spikes are sent back to neuron σ_r at step $t+5$ from neurons $\sigma_{l_i^6}, \sigma_{l_i^7}, \ldots, \sigma_{l_i^{10}}$, which represents that the number in register r is still zero.

The simulation of SUB instruction is correct: system Π' starts from neuron σ_{l_i} and ends in neuron σ_{l_j} (if the number stored in register r is great than 0), or in neuron σ_{l_k} (if the number stored in register r is 0).

Table 3 The numbers of spikes in the neurons of SUB module during the SUB instruction simulation with the case that neuron σ_r has $n+5$ ($n>0$) spikes

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	$n+5$	$n+4$	$n+4$	$n+4$	$n+4$	$n+4$
$\sigma_{l_i^1}$	0	1	0	0	0	0
$\sigma_{l_i^2}$	0	-1	-1	-1	-1	0
$\sigma_{l_i^3}$	0	0	0	0	0	0
$\sigma_{l_i^4}$	0	0	0	0	0	0
$\sigma_{l_i^5}$	0	0	0	0	0	0
$\sigma_{l_i^6}$	0	0	0	0	0	0
$\sigma_{l_i^7}$	0	0	0	0	0	0
$\sigma_{l_i^8}$	0	0	0	0	0	0
$\sigma_{l_i^9}$	0	0	0	0	0	0
$\sigma_{l_i^{10}}$	0	0	0	0	0	0
$\sigma_{l_i^{11}}$	0	0	1	0	0	0
$\sigma_{l_i^{12}}$	0	0	0	1	0	0
$\sigma_{l_i^{13}}$	0	0	0	0	1	0
σ_{l_j}	0	0	0	0	0	1
σ_{l_k}	0	0	0	0	0	0

If several SUB instructions l_v acting on register r, then neuron σ_r may send one spike to all the auxiliary neurons $\sigma_{l_v^2}, \sigma_{l_v^3}$ with $l_v \neq l_i$, when we simulate instruction l_i at certain step. One step later, by consuming the spike inside, neurons $\sigma_{l_v^2}$ and $\sigma_{l_v^3}$ fire sending a spike to neuron $\sigma_{l_i^4}$, respectively. The spike from neuron $\sigma_{l_v^2}$ will be changed into an anti-spike when it arrives at neuron $\sigma_{l_v^4}$. So, in neuron $\sigma_{l_v^4}$, there are a spike and an anti-spike. After the annihilation, neuron $\sigma_{l_v^4}$ returns to the state that there is neither spike nor anti-spike inside. Consequently, the interference among SUB modules will not cause wrong steps in Π'.

The evolution of the numbers of spikes of SUB module during the SUB instruction simulation is shown in Tables 3 and 4, corresponding to the two cases in which, at step t, the number stored in register r is n, $n>0$ and the number stored in register r is 0, respectively.

Module FIN shown in Fig. 4—outputting the result of computation.

Assume that the number in register r is n, i.e., there are $n+5$ spikes in neuron σ_r. At certain step t, neuron σ_{l_h} fires sending one spike to neurons σ_{h_1}, σ_{h_3} and σ_{h_4}. The first spike arriving in neuron σ_{out} at step $t+2$, so neuron σ_{out} fires for the first time at step $t+3$ emitting out one spike to the environment.

Moreover, at step $t+2$, neuron σ_1 receives the first anti-spike, and from that moment on, an anti-spike is continuously sent to neuron σ_1 in each step (because neurons $\sigma_{l_h^3}$ and $\sigma_{l_h^5}$ continuously exchange one spike among them). Until step $t+n+2$, neuron σ_1 ends with 4 spikes (since it totally receives $n+1$ anti-spikes). and fires sending one spike to neuron σ_{out}. At step $t+n+3$, neuron σ_{out} fires for the second time and emits the second spike out to the environment. The time interval of the first two spikes emitted by neuron σ_{out} is $(t+n+3)-(t+3)=n$, which is the number stored in register 1. When neuron σ_1 fires, it also sends one spike to neurons σ_{h_3} and σ_{h_5}. But neurons σ_{h_3} and σ_{h_5} will receive an

Table 4 The numbers of spikes in the neurons of SUB module during the SUB instruction simulation with the case that neuron σ_r has 5 ($n = 0$) spikes

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	5	4	0	0	0	5
$\sigma_{l_i^1}$	0	1	0	0	0	0
$\sigma_{l_i^2}$	0	−1	0	0	0	0
$\sigma_{l_i^3}$	0	0	1	0	0	0
$\sigma_{l_i^4}$	0	0	0	1	0	0
$\sigma_{l_i^5}$	0	0	0	0	1	0
$\sigma_{l_i^6}$	0	0	0	0	1	0
$\sigma_{l_i^7}$	0	0	0	0	1	0
$\sigma_{l_i^8}$	0	0	0	0	1	0
$\sigma_{l_i^9}$	0	0	0	0	1	0
$\sigma_{l_i^{10}}$	0	0	0	0	1	0
$\sigma_{l_i^{11}}$	0	0	1	0	0	0
$\sigma_{l_i^{12}}$	0	0	0	1	0	0
$\sigma_{l_i^{13}}$	0	0	0	0	0	0
σ_{l_j}	0	0	0	0	0	0
σ_{l_k}	0	0	0	0	0	1

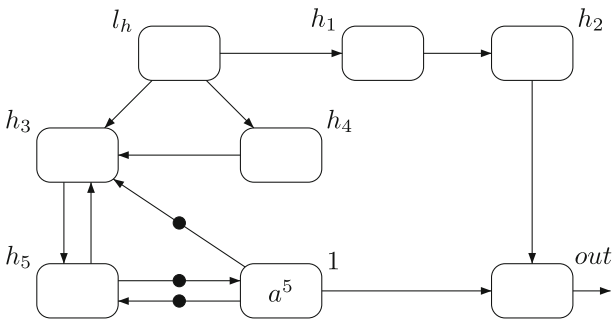

Fig. 4 Module FIN of Π (outputting the result of the computation)

anti-spike due to the inhibitory synapses $(1, h_3)$ and $(1, h_5)$. After the annihilation, neurons σ_{h_3} and σ_{h_5} stop to exchange the spike among them, hence the system halts.

The evolution of the numbers of spikes in the neurons during the computation of the FIN module is shown in Table 5.

From the above description of the modules and their works, it is clear that register machine M can be correctly simulated by HISN P system Π, i.e. the HISN P systems can compute any set of natural numbers, which is computable by register machines.

Therefore, HISN P systems can characterize NRE. We can check that the homogeneous rule set of Π contains only one category of spiking rules identified by (a, a), and there are only 5 spiking rules in each neuron. □

Table 5 The numbers of spikes in neurons of FIN module during the process outputting the computational result

Neuron	Step						
	t	$t+1$	$t+2$	$t+3$	...	$t+n+2$	$t+n+3$
σ_{l_h}	1	0	0	0	...	0	0
σ_{h_1}	0	1	0	0	...	0	0
σ_{h_2}	0	0	1	0	...	0	0
σ_{h_3}	0	1	1	1	...	1	0
σ_{h_4}	0	1	0	0	...	0	0
σ_{h_5}	0	0	1	1	...	1	0
σ_1	$n+5$	$n+5$	$n+5$	$n+4$	...	4	0
σ_{out}	0	0	0	1	...	0	1

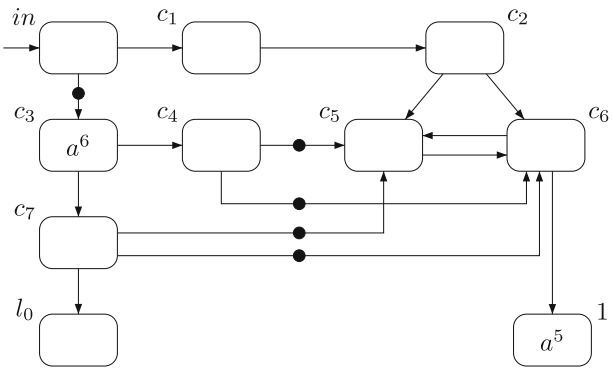

Fig. 5 The INPUT module

HISN P systems working in accepting mode can accept any set of Turing computable natural numbers. This result is achieved by simulating register machines working in acceptive mode. In the simulation, the FIN module is not necessary any more, but an INPUT module is needed.

Theorem 3.2 $N_{acc}HISNP(cate_1, rule_5, Inh) = NRE$ *with* $cate_1 = \{(a, a)\}$.

Proof Let M be a register machine working in the acceptive mode. An HISN P system Π' composed of neurons shown in Fig. 1 is constructed to simulate M. Each register r of M is associated with a neuron σ_r in Π', and if the number stored in register r is n, neuron σ_r will contain $n + 5$ spikes.

The system Π' consists of deterministic ADD modules, SUB modules, and an INPUT module. The INPUT module is shown in Fig. 5, where all neurons are initially empty with the exception of neurons σ_1 and σ_{c_3}, which contain 5 and 6 spikes, respectively. Spike train $10^{n-1}1$ is introduced in the system from the environment through the input neuron σ_{in}. It indicates that number n is going to be accepted by the system.

At step 1, neuron σ_{in} receives one spike from the environment. It fires one step later sending one spike to neurons σ_{c_1} and σ_{c_3}. When the spike arrives at neuron σ_{c_3}, it will be changed into an anti-spike, so one of the spikes in neuron σ_{c_3} will be annihilated. Neuron σ_{c_3} keeps inactive before it receives the second anti-spike. At step 5, neurons σ_{c_5} and σ_{c_6} fire by receiving one spike from neuron σ_{c_2}. From that moment on, the two neurons begin to

Table 6 The numbers of spikes in the neurons of INPUT module during the process of reading spike train $10^{n-1}1$

Neuron	Step									
	1	2	3	4	5	...	$n+1$	$n+2$	$n+3$	$n+4$
σ_{in}	1	0	0	0	0	...	1	0	0	0
σ_{c_1}	0	1	0	0	0	...	0	1	0	0
σ_{c_2}	0	0	1	0	0	...	0	0	1	0
σ_{c_3}	6	5	5	5	5	...	5	4	0	0
σ_{c_4}	0	0	0	0	0	...	0	0	1	0
σ_{c_5}	0	0	0	1	1	...	1	1	1	0
σ_{c_6}	0	0	0	1	1	...	1	1	1	0
σ_{c_7}	0	0	0	0	0	...	0	0	1	0
σ_{l_0}	0	0	0	0	0	...	0	0	0	1
σ_1	5	5	5	5	6	...	$n+2$	$n+3$	$n+4$	$n+5$

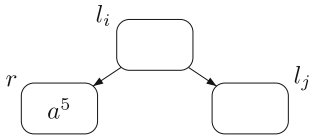

Fig. 6 The deterministic ADD module of Π'

exchange one spike among them, and neuron σ_{c_6} continuously sends one spike to neuron σ_1 in each step.

At step $n+1$, neuron σ_{in} receives the second spike from the environment (the time internal of the two spikes entering in neuron σ_{in} is $(n+1)-1=n$). It fires at that time sending one spike to neurons σ_{c_1} and σ_{c_3} again. Neuron σ_{c_3} receives the second anti-spike, and after the annihilation, it ends with four spikes. Neuron σ_{c_3} fires at step $n+3$, sending one spike to neurons σ_{c_4} and σ_{c_7}. The two neurons will send two spikes to neurons σ_{c_5} and σ_{c_6} along inhibitory synapses $(\sigma_{c_4}, \sigma_{c_5}, (\sigma_{c_4}, \sigma_{c_6}), (\sigma_{c_7}, \sigma_{c_5})$ and $(\sigma_{c_7}, \sigma_{c_6})$. Hence, neurons σ_{c_5} and σ_{c_6} will receive two anti-spikes at step $n+4$. This stops neuron σ_{c_7} to send spikes to neuron σ_1. From step 5 to $n+4$, neuron σ_1 totally receives n spikes, that is, there are $n+5$ spikes in neuron σ_1. At step $n+4$, neuron σ_{l_0} receives one spike from neuron σ_{c_7}. It fires in the next step to start to simulate the initial instruction l_0 of M.

During the process of reading spike train $10^{n-1}1$, the evolution of the numbers of spikes in the neurons of INPUT module is shown in Table 6.

The deterministic ADD module simulating the deterministic ADD instruction is shown in Fig. 6. Module SUB remains unchanged (as shown in Fig. 3), module FIN is removed, with neuron σ_{l_h} remaining in the system, but without outgoing synapses. When neuron σ_{l_h} receives one spike, it means that the computation of register machine M reaches instruction l_h and stops. Having one spike inside, neuron σ_{l_h} fires without emitting any spike out. The system Π' halts. As explained above, it is obtained that any set of natural numbers accepted by deterministic register machines can be accepted by HISN P system Π'.

In the system Π', the number n to be computed is introduced through the input neuron and stored in neuron σ_1 in the form of $n+5$ spikes. We can check that a computation of HISN P system Π' stops if and only if the computation of M stops. The system is composed

of neuron in Fig. 1, that is, each neuron contains one category of spiking rules, and there are 5 spiking rules in the neuron. □

4 Concluding Remarks

In this work, we consider a restricted variant of SN P systems with inhibitory synapses, called homogenous SN P systems with inhibitory synapses. In the systems, each neuron has the same set of spiking and forgetting rules in sense of having the homogeneity. We investigate the computational power of the systems. It is obtained that such systems with using one category of spiking rule and no forgetting rule can achieve Turing completeness, i.e., compute and accept any set of Turing computable natural numbers.

In the proofs of Theorem 3.1 and Theorem 3.2, the neurons in the systems contain 5 spiking rules of category (a, a). It is worth to consider that whether we can reduce the number of rules in the neuron by using the other three categories of spiking rules (identified by $(a, \bar{a})$, $(\bar{a}, a)$ and $(\bar{a}, \bar{a})$) without losing the universality of HISN P systems.

Asynchronous SN P systems were proposed in [38], where each neuron with enabled rules is not obligatory to use one rule, that it, the neuron is free to choose whether to use a rule or not. This feature is closer to the way of biological neuron processing information. It is interesting to investigate if the type of neurons in universal asynchronous SN P systems can be reduced by using inhibitory synapses.

In neural systems, some properties, such as variability of delay times, synaptic failure, adaptivity, refractoriness, Dale's law are elemental to the computability of the systems. Can we simplify the structure of universal SN P systems, such as having less number of neurons and synapses by using these properties?

SN P systems with anti-spikes and SN P systems with inhibitory synapse work in different manners. In SN P systems with anti-spikes, a neuron can produce anti-spikes and send them to each of the neighboring neurons. In SN P systems with inhibitory synapses, the spike can be changed into anti-spike only when they pass along the inhibitory synapses. The structure and neuron of homogenous universal SN P systems with inhibitory synapses obtained in this work are simpler than homogenous universal SN P systems with anti-spikes have been proposed in [39]. It remains open that if it is possible to further simplify the structure and neuron in universal SN P systems by using both inhibitory synapses and anti-spikes.

In biological neural systems, there are 70 % of pyramidal cell, while the remain 30 % are or a number of different types. The inhibitory neurons by no means similar. It is interesting to consider SN P systems with heterogeneous inhibition, as well as investigate the computational power of the systems.

The communication mechanism of cell-like and tissue-like P systems has provide rich source to design intelligent algorithms. The obtained algorithms are usually called membrane algorithms or membrane-inspired algorithms [40], which have been used in some other fields, such as solving computational hard problems [5,6], image processing [41], industrial engineering [42,43], designing DNA sequences in DNA computing [44]. The framework of SN P systems is inspired by the human brain, which is known as the most powerful computing "device" in natural. How to design intelligent algorithms using SN P systems and explore their potential applications are worth for further research.

Acknowledgments This work was supported by National Natural Science Foundation of China (61033003, 91130034, 61100145, 61202011 and 61272071), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072 and 2012014213 008), and Natural Science Foundation of Hubei Province (2011CDA027).

References

1. Rozenberg G, Bck T, Kok JN (2011) Handbook of natural computing. Springer, Berlin
2. Păun G (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143
3. Freund R, Păun G, Pérez-Jiménez MJ (2005) Tissue-like P systems with channel-states. Theor Comput Sci 330:101–116
4. Ionescu M, Păun G, Yokomori T (2006) Spiking neural P systems. Fund Inf 71(2–3):279–308
5. Nishida TY (2004) An approximate algorithm for NP-complete optimization problems exploiting P systems. In Proceedings of Brainstorming Workshop on Uncertainty in Membrane, Computing, 185–192
6. Zhang G, Cheng J, Gheorghe M, Meng Q (2013) A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. Appl Soft Comput 13(3):1528–1542
7. Păun G, Rozenberg G, Salomaa A (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford
8. Siegelmann HT, Sontag ED (1995) On the computational power of neural nets. J Comput System Sci 50(1):132–150
9. Pan L, Zeng X (2010) A note on small universal spiking neural P systems. LNCS 5957:436–447
10. Pollack JB (1991) The induction of dynamical recognizers. Mach Learn 7(2–3):227–252
11. Giles CL, Omlin CW (1992), Inserting rules into recurrent neural networks. Neural Networks for Signal Processing II. In Proceedings of the Signal Processing, Workshop, 13–22
12. Hyoetyniemi H (1996) Turing machines are recurrent neural networks. In Proceedings of Genes, Nets And Symbols. Helsinki, Finland: Finnish Artificial Intelligence Society, 13–24
13. Moore C (1998) Dynamical recognizers: real-time language recognition by analog computers. Theor Comput Sci 201:99–136
14. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10:1659–1671
15. Maass W, Schmitt M (1999) On the complexity of learning for spiking neurons with temporal coding. Inf Comput 153(1):26–46
16. Natschläger T, Maass W (2002) Spiking neurons and the induction of finite state machines. Theor Comput Sci 287(1):251–265
17. Ghosh DS, Adeli H (2009) Spiking neural networks. Int J Neural Syst 19(4):295–308
18. Pan L, Păun G (2010) Spiking neural P systems: an improved normal form. Theor Comput Sci 411(6):906–918
19. Zeng X, Pan L (2009) Homogeneous spiking neural P systems. Fund Inf 97:275–294
20. Chen H, Freund R, Ionescu M et al (2007) On string languages generated by spiking neural P systems. Fund Inf 75(1–4):141–162
21. Zhang X, Zeng X, Pan L (2008) On string languages generated by SN P systems with exhaustive use of rules. Nat Comput 7(4):535–549
22. Pan L, Zeng X (2011) Small universal spiking neural P systems working in exhaustive mode. IEEE Trans Nanobiosci 10(2):99–105
23. Păun A, Păun G (2007) Small universal spiking neural P systems. Biosystems 90:48–60
24. Ishdorj TO, Leporati A, Pan L et al (2010) Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Theor Comput Sci 411(25):2345–2358
25. Pan L, Păun G, Pérez-Jiménez MJ (2011) Spiking neural P systems with neuron division and budding. Sci China Inform Sci 54(8):1596–1607
26. Ionescu M, Sburlan D (2007) Some applications of spiking neural P systems. In Proceedings of the Eighth Workshop on Membrane Computing, Thessaloniki, 383–394
27. Cavaliere M, Ibarra OH, Păun G (2009) Asynchronous spiking neural P systems. Theor Comput Sci 410:2352–2364
28. Song T, Pan L, Păun G (2013) Asynchronous spiking neural P systems with local synchronization. Inf Sci 219:197–207
29. Pan L, Wang J, Hoogeboom HJ (2012) Spiking neural P systems with astrocytes. Neural Comput 24:1–24
30. Macías-Ramos LF, Pérez-Jiménez MJ (2013) Spiking neural P systems with functional astrocytes. LNCS 7762:228–242

31. Păun G (2007) Spiking neural P systems with astrocyte-like control. J Univers Comput Sci 13(11):1707–1721
32. Song T, Pan L (2014) Spiking neural P systems with rules on synapses. Theor Comput Sci 529:82–95
33. Pan L, Păun G (2009) Spiking neural P systems with anti-spikes. Int J Comput Commun 4(3):273–282
34. Song T, Pan L, Wang J et al (2012) Normal forms of spiking neural P systems with anti-spikes. IEEE Trans Nanobiosci 4(11):352–359
35. Rozenberg G, Salomaa A (1997) Handbook of formal languages. Springer, Berlin
36. Minsky M (1967) Computation: finite and infinite machines. Prentice Hall, Upper Saddle River
37. Păun G (2002) Membrane computing: an introduction. Springer, Berlin
38. Cavaliere M, Egecioglu O, Ibarra OH et al (2009) Asynchronous spiking neural P systems. Theor Comput Sci 410:2352–2364
39. Song T, Wang X, Zhang Z et al (2013) Homogenous spiking neural P systems with anti-spikes. Neural Comput Appl. doi:10.1007/s00521-013-1397-8
40. Nishida TY (2006) Membrane algorithms: approximate algorithms for np-complete optimization problems. Applications of membrane computing. Springer, Berlin
41. Zhang G, Gheorghe M, Li Y (2012) A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. Nat Comput 11(4):701–717
42. Zhao J, Wang N (2011) Hybrid optimization method based on membrane computing. Ind Eng Chem Res 50(3):1691–1704
43. Wang K, Wang N (2011) A protein inspired RNA genetic algorithm for parameter estimation in hydrocracking of heavy oil. Chem Eng J 167(1):228–239
44. Xiao J, Zhang X, Xu J (2012) A membrane evolutionary algorithm for DNA sequence design in DNA computing. Chin Sci Bull 57(1):698–706