Neural Comput & Applic
DOI 10.1007/s00521-013-1397-8

ORIGINAL ARTICLE

Homogenous spiking neural P systems with anti-spikes

Tao Song · Xun Wang · Zhujin Zhang · Zhihua Chen

Received: 16 December 2012 / Accepted: 23 March 2013

Abstract Spiking neural P systems with anti-spikes (ASN P systems, for short) are a class of neural-like computing models in membrane computing, which are inspired by neurons communication through both excitatory and inhibitory impulses (spikes). In this work, we consider a restricted variant of ASN P systems, called homogeneous ASN P systems, where any neuron has the same set of spiking and forgetting rules. As a result, we prove that such systems can achieve Turing completeness. Specifically, it is proved that two categories of pure form of spiking rules (for a spiking rule, if the language corresponding to the regular expression that controls its application is exactly the form of spikes consumed by the rule, then the rule is called pure) are sufficient to compute and accept the family of sets of Turing computable natural numbers.

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

T. Song · Z. Chen
Department of Control Science and Engineering,
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

Z. Zhang
Department of Computer Science, Shenzhen Graduate School,
Harbin Institute of Technology, Shenzhen 518055, China

1 Introduction

In neural networks of human brains, the structure and functioning of "information processing mechanism" are quite intricate, which provide a rich source of inspiration for computer scientists to design computational devices and intelligent machines. Concretely, neural-inspired computing models based on the concept of spiking neurons are a new branch of natural computing and have been heavily investigated [9, 10]. In 2006, a new class of neural computing devices, called spiking neural P systems (SN P systems, for short), were introduced within the framework of membrane computing, which is one of the most recent branches of natural computing. Membrane computing was initiated in [21], and SN P systems are now known as a specific class of neural-like models inspired by the way neurons communicate with each other. A 2009 compiled bibliography on SN P systems can be found in [22], with additional material also available in the corresponding chapter of [23].

Generally, an SN P system can be graphically represented by a directed graph, where neurons are placed in the nodes sending and receiving signals along synapses represented by the directed arcs. Every neuron in the systems may contain a number of spikes, spiking rules, and forgetting rules. By applying spiking rules, the information in a certain neuron can be processed and sent to its target neurons in the form of spikes. When a neuron uses a forgetting rule, a specified number of spikes are removed out of the neuron. The applicability of each rule is determined by checking the content (number of spikes) of the neuron. SN P systems usually work in the synchronous manner. A global clock marks the time of the whole system, and if at any step a rule can be applied in any neuron, that rule must be applied (if there are more than one enabled rules in the

Published online: 01 June 2013

neuron, then one of them will be non-deterministically chosen and applied), but for different neurons, they work in parallel.

One neuron is distinguished as the output neuron which emits spikes to the environment. The time interval between the first two spikes sent out to the environment by the output neuron contributes to the result of the system computation.

SN P systems have been proved to be Turing complete, that is, they can do what a Turing machine does [1, 4, 20]. In previous works, SN P systems are used as computing devices mainly in three ways: generating/computing sets of numbers [4, 14, 26], generating languages [1, 27] and computing functions [17, 20]. SN P systems with budding and cell division were able to (theoretically) generate exponential working space in linear time, thus providing a way to solve **NP**-complete problems in feasible time (see, e.g., [6, 15]). Other applications of SN P systems can be found in [5]. Inspired from different biological facts in biological neural networks, many variants of SN P systems have been investigated, such as SN P systems with weighted synapses [19], SN P systems with astrocytes [16], time-free SN P systems [18] and asynchronous SN P systems with local synchronization [24].

Furthermore, many researches focus on simulating these SN P systems. In [11], P-Lingua was developed to simulate many variants of SN P systems, as well as some hardware-based SN P systems simulators were also designed in [2, 3].

This work theoretically deals with another variant of SN P systems, called SN P systems with anti-spikes (ASN P systems, for short), which are inspired by the way neurons communicate by both excitatory and inhibitory spikes [13]. In these systems, besides using spikes, anti-spikes are also considered. In ASN P systems, a neuron can generate spikes or anti-spikes by using spiking rules and send them to the target neurons. Also, by using forgetting rules, a specified number of spikes or anti-spikes (not both of them) will be removed from the neuron. In each neuron, the spikes and anti-spikes can participate in an annihilating rule, that is, when spikes and anti-spikes meet in a certain neuron, they will immediately annihilate each other in a maximal manner. Hence, at any moment, there are only spikes or anti-spikes in any neuron but not both of them. One output neuron is specified to emit spikes to the environment, but it cannot emit anti-spikes. The time interval between the first two spikes emitted to the environment by the output neuron can be defined as the computation result. It was proved that ASN P systems can generate any set of Turing computable natural numbers [13].

Considering homogeneity in ASN P systems has a clear biological motivation. In the (human) brain, there exist about 100 billion neurons, and most of them are very similar to each other [7, 25]. They are "designed" by nature to have the same "set of rules," "working" in a uniform way to transform input into output. In this work, a restricted variant of ASN P systems, called homogenous ASN P systems (shortly called HASN P systems), is investigated, where all the neurons in HASN P systems have the same set of rules, that is having homogeneity. In the usual ASN P systems, neurons are "different" from each other, that is having different rules. So, from this point of view, HASN P systems are closer to biological reality than the usual ASN P systems, and in this sense, the computing power of such systems investigated in this paper is interesting.

As a result, we prove that HASN P systems not using forgetting rules in any neuron can achieve the Turing completeness. Specifically, HASN P systems can compute and accept the family of sets of Turing computable numbers by using only two categories of pure form of spiking rules. These results provide a class of practically feasible computing models, since ASN P systems with only pure form of rules can avoid the computationally hard problem hidden in checking whether a non-pure spiking rule can be applied or not [8]. Those universality results can have a good interpretation: the structure of a neural system is crucial for the functioning of the system. Although the individual neuron is simple and homogeneous, by cooperating with each other, a network of neurons can be rather powerful.

The structure of this paper is as follows. Section 2 introduces the notion of homogeneous SN P systems with anti-spikes, and the computing power of the systems is discussed in Sect. 3. Finally, some open problems are given in Sect. 4.

2 Homogenous spiking neural P systems with anti-spikes

In this section, we introduce the formal definition of HASN P system, as well as the family of sets of numbers generated (computed) and accepted by the systems. The definition is complete, but familiarity with the basic elements of classic SN P systems (e.g., from [4]) is advisable.

An *HASN 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 called *spike* and $\bar{a}$ is called *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. $n_i \in \mathbb{Z}$ is the *initial number of spikes* contained in neuron σ_i;
2. R is the homogenous set of *rules* of the following two forms:
   (a) $E/b^c \rightarrow b'$, where E is the regular expression over $\{a\}$ or $\{\bar{a}\}, b, b' \in \{a, \bar{a}\}$ and $c \geq 1$;
   (b) $b^s \rightarrow \lambda$ for some $s \geq 1$ with the restriction that $b^s \notin L(E)$ for any rule $E/b^c \rightarrow b'$ from R and $b \in \{a, \bar{a}\}$;

– $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ is the set of *synapses* between neurons;
– 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 to the environment.

In HASN P systems, every neuron has the same set of rules, that is, having the homogeneity. The rules of the form $E/b^c \rightarrow b'$ are *spiking rules*. They are applied as follows: for a neuron σ_i, if it contains k spikes/anti-spikes b with $b^k \in L(E)$ and $k \geq c$, then the rule $E/b^c \rightarrow b'$ can be applied. By using the rule, c spikes/anti-spikes are consumed (thus $k - c$ spikes/anti-spikes remain in the neuron) and one spike/anti-spike is sent out to all neurons σ_j such that $(i, j) \in syn$. There are four categories of spiking rules identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$. For any spiking rule $E/b^c \rightarrow b'$, if $L(E) = b^c$, it is simply written as $b^c \rightarrow b'$ and called pure. 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 can be used in a neuron at some moment, while only one of them is chosen non-deterministically. It is important to notice that the applicability of a spiking rule is controlled by checking the number of spikes/anti-spikes contained in the neuron against a regular expression over $\{a\}$ or $\{\bar{a}\}$ associated with the rule.

Rules of the form $b^s \rightarrow \lambda$ with $b \in \{a, \bar{a}\}$ and $s \geq 1$ are *forgetting rules*, by which a specified number of spikes/anti-spikes can be removed out of the neuron. For a neuron σ_i, if it contains exactly s spikes/anti-spikes, then the forgetting rule $b^s \rightarrow \lambda, b \in \{a, \bar{a}\}$ from R can be applied. For any regular expression E associated with a spiking rule from R, if it satisfies that $b^s \notin L(E)$ meaning that a spiking rule is applicable, then no forgetting rule is applicable, and vice versa.

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.

When spikes and anti-spikes meet in a neuron, they will immediately annihilate each other in a maximal manner. Specifically, if a neuron contains a^r spikes and $\bar{a}^s$ anti-spikes inside, then the annihilating rule $a\bar{a} \rightarrow \lambda$ is immediately applied in a maximal manner, ending with $r - s$ spikes or $s - r$ anti-spikes in the neuron. Hence, at any moment, each neuron in the system can have either spikes or anti-spikes, but not both of them. It is this reason why the regular expression E from any spiking rule $E/b^c \rightarrow b'$ is over $\{a\}$ or $\{\bar{a}\}$, but not over $\{a, \bar{a}\}$.

The configuration of HASN P systems is of the form $\langle c_1, c_2, \ldots, c_m \rangle$ with $c_i \in \mathbb{Z}$. At any moment, if $c_i \geq 0$, it means that there are c_i "positive" spikes in neuron σ_i; if $c_i < 0$, it indicates that neuron σ_i contains c_i anti-spikes. By using spiking and forgetting rules, one can define *transitions* among configurations. A series of transitions starting from the initial configuration is called a *computation*. The computation halts if it proceeds to a configuration such that no rules can be applied in any neuron. With any computation, halting or not, we associate a *spike train*, a sequence of natural numbers $t_1, t_2, \ldots$ with $1 \leq t_1 < t_2 < \ldots$, indicating time instances when the output neuron sends a spike out of the system. For any spike train containing at least two spikes, the distance of first two spikes being emitted at steps t_1,t_2 is considered as the computation result, in the form of number $t_2 - t_1$; we say that this number is computed by system 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 HASN P system Π can also work in the accepting mode. A number n is stored in a specified neuron in the form of $f(n)$ spikes, which are introduced in the system in form of spike train 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 $N_\alpha HASNP(cate_l, rule_k, forg_m)$, $\alpha \in \{2, acc\}$ all sets of numbers generated or accepted by HASN P systems with at most l categories of spiking rules, at most k rules in each neuron, and forgetting rules removing at most m spikes each time. If only pure form of spiking rules is applied, the indication $rule_k$ is substituted by $prule_k$, and if the forgetting rules are not used in any neuron, the indication of $forg_m$ will be removed from the notation.

3 The computing power of HASN P systems

In this section, we prove that HASN P systems with no forgetting rule can generate and accept any set of Turing computable natural numbers.

The following proofs are based on the simulation of register machines. A register machine is a construct

$M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label, and I is the set of instructions; each label from H labels only one instruction from I. The instructions are of the following three forms:

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

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

It is known that register machines compute and accept (using deterministic ADD instructions) all sets of numbers which are Turing computable, hence they characterize *NRE*, that is the family of Turing computable sets of numbers (see, e.g., [12]).

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 the power of two number generating/accepting devices are compared, number zero is ignored. This corresponds to the practice of ignoring the empty string when comparing the power of two devices in language and automata theory.

In the proofs, HASN P systems are represented graphically, which is easier to understand than in a symbolic way. We use an oval and initial number of spikes to represent a neuron (since any neuron has the same set of rules, we do not indicate the set of rules in each neuron), and a directed graph to represent the structure of the system: the neurons are placed in the nodes of the graph and the edges represent the synapses; the input/output neuron has an inputting/outgoing arrow, suggesting their communication with the environment.

Theorem 1 $N_2HASNP(cate_2,prule_6) = NRE$ *with* $cate_2 = \{(a, a), (a, \bar{a})\}$.

Proof It is enough to prove the inclusion $NRE \subseteq N_2HASNP(cate_2, prule_6)$, for the converse inclusion can be invoked from the Turing-Church thesis. In the following universality proof, an HASN P system Π is constructed to simulate the computation of M. Each register r of M is represented by a neuron σ_r, whose content corresponds to the number stored in register r. Specifically, if the number stored in register r is $n \geq 0$, then neuron σ_r contains $n + 6$ spikes. In the initial configuration, the number stored in each register of M is 0, so that there are six spikes in neurons representing those registers. One neuron σ_{l_i} is associated with each instruction labeled l_i of M. During a computation, a neuron σ_{l_i} having one spike inside will become active and start to simulate an instruction $l_i : (\texttt{OP}(r), l_j, l_k)$ of M ($\texttt{OP} \in \{\texttt{ADD}, \texttt{SUB}\}$): starting with neuron σ_{l_i} activated, operating neuron σ_r as requested by OP, then introducing one spike into neuron σ_{l_j} or neuron σ_{l_k}, which becomes active in this way. When neuron σ_{l_h} (associated with the label l_h of the halting instruction of M) is activated, a computation in M is completely simulated in Π; the FIN module starts to output the computation result (the time internal between the first two spikes emitted by the output neuron corresponds to the number stored in register 1 of M).

The HASN P system Π is composed of three types of modules, ADD modules, SUB modules, and a FIN module shown in Figs. 2, 3 and 4, which are used to simulate the ADD, SUB, and HALT instructions of M. Each module is composed of neurons from Fig. 1 with homogenous set of rules $R = \{a \to a, a^2 \to a, a^3 \to \bar{a}, a^4 \to a, a^4 \to \bar{a}, a^5 \to a\}$.

Module ADD shown in Fig. 2—simulating the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

The initial instruction of M, the one with label l_0, 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 inside, neuron σ_{l_i} fires at step t by using the rule $a \to a$. Immediately, one spike is sent to neuron σ_r from σ_{l_i}. The number of spikes in neuron σ_r is increased by one, which simulates the number in register r is increased by one. At the same time, neuron σ_{l_i} sends one spike to neurons $\sigma_{l_i^1}, \sigma_{l_i^2}, \sigma_{l_i^3}$ and $\sigma_{l_i^4}$. One step later, the four neurons become active sending four spikes to neuron $\sigma_{l_i^7}$, while each of neurons $\sigma_{l_i^1}$ and $\sigma_{l_i^2}$ sends one spike to neuron $\sigma_{l_i^5}$ and $\sigma_{l_i^6}$, respectively, one spike to neurons $\sigma_{l_i^5}$ and $\sigma_{l_i^6}$, respectively. With four spikes inside, neuron $\sigma_{l_i^7}$ fires at step $t + 2$ by

Fig. 1 The neuron in system Π

$$a \to a$$
$$a^2 \to a$$
$$a^3 \to \bar{a}$$
$$a^4 \to a$$
$$a^4 \to \bar{a}$$
$$a^5 \to a$$

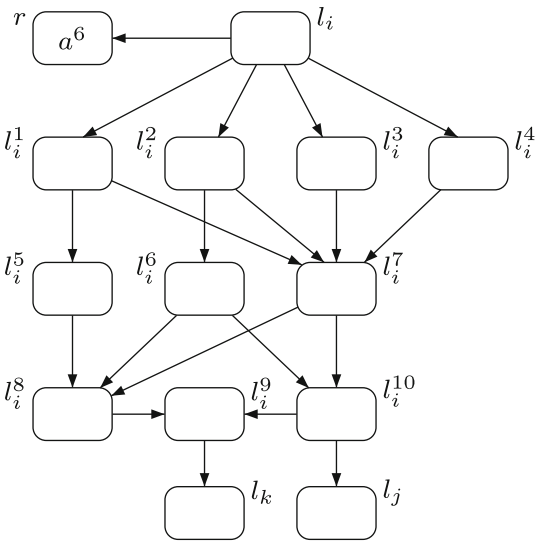

Fig. 2 Module ADD of Π (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

non-deterministically using one of the two spiking rules $a^4 \rightarrow a$ and $a^4 \rightarrow \bar{a}$. This will non-deterministically activate neuron σ_{l_j} or σ_{l_k}.

At step $t + 2$, if neuron $\sigma_{l_i^7}$ uses the rule $a^4 \rightarrow a$, it sends one spike to neurons $\sigma_{l_i^8}$ and $\sigma_{l_i^{10}}$. Meanwhile, neurons $\sigma_{l_i^5}$ and $\sigma_{l_i^6}$ fire sending one spike to neuron $\sigma_{l_i^{10}}$, as well as two spikes to neuron $\sigma_{l_i^8}$. Having two spikes inside, neuron $\sigma_{l_i^{10}}$ fires at step $t + 3$ and sends one spike to neurons $\sigma_{l_i^9}$ and σ_{l_j}. At the same step, neuron $\sigma_{l_i^8}$ sends an anti-spike neuron $\sigma_{l_i^9}$, which annihilates the spike in the neuron immediately. So, neuron $\sigma_{l_i^9}$ ends with no spike or anti-spike inside and remains inactive. By receiving the spike, neuron σ_{l_j} fires at step $t + 4$, simulating instruction l_j of M.

At step $t + 2$, if neuron $\sigma_{l_i^7}$ fires by using the rule $a^4 \rightarrow \bar{a}$, it sends an anti-spike to neurons $\sigma_{l_i^8}$ and $\sigma_{l_i^{10}}$. Meanwhile, neuron $\sigma_{l_i^6}$ sends a spike to neuron $\sigma_{l_i^{10}}$. After the annihilation, neuron $\sigma_{l_i^{10}}$ ends with no spike or anti-spike inside and cannot fire. At step $t + 2$, besides receiving the anti-spike from neuron $\sigma_{l_i^7}$, neuron $\sigma_{l_i^8}$ also receives two spikes from neurons $\sigma_{l_i^5}$ and $\sigma_{l_i^6}$. Neuron $\sigma_{l_i^8}$ ends with one spike inside and can fire by using the rule $a \rightarrow a$ one step later. At step $t + 3$, neuron $\sigma_{l_i^8}$ sends one spike to neuron $\sigma_{l_i^9}$, which becomes active one step later emitting one spike to neuron σ_{l_k}. At step $t + 5$, neuron σ_{l_k} fires, simulating instruction l_k of M.

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

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

Assuming at certain step t, the system starts to simulate a SUB instruction of M. Having one spike inside, neuron σ_{l_i} fires at step t sending a spike to neurons $\sigma_{l_i^1}, \sigma_{l_i^2}$ and $\sigma_{l_i^3}$. The three neurons become active at step $t + 1$ and send three spikes to neuron $\sigma_{l_i^4}$. With three spikes, neuron $\sigma_{l_i^4}$ fires at step $t + 2$ sending an anti-spike to neurons $\sigma_{l_i^5}$ and σ_r. In neuron σ_r, we have the following two cases.

1. Neuron σ_r has $n + 6$ spikes (corresponding to the fact that the number stored in register r is n, $n > 0$) before it receives the anti-spike from neuron σ_{l_i}. By receiving the anti-spike from neuron $\sigma_{l_i^4}$, one spike in neuron σ_r is immediately annihilated. The number of spikes in neuron σ_r is decreased by one to simulate that the number stored in register r decreases by one. Neuron σ_r will end with at least six spikes and cannot fire since there is no rule to use. The spike in neuron $\sigma_{l_i^5}$ will be annihilated by the anti-spike from neuron $\sigma_{l_i^4}$. In this way, neuron $\sigma_{l_i^5}$ remains inactive. Moreover, one spike arrives in neuron $\sigma_{l_i^{23}}$ at step $t + 8$ by passing along path $\sigma_{l_i}, \sigma_{l_i^1}, \sigma_{l_i^{16}}, \ldots, \sigma_{l_i^{22}}$. One step later, neuron $\sigma_{l_i^{23}}$ becomes active sending one spike to neurons $\sigma_{l_i^5}$ and σ_{l_j}. In neuron $\sigma_{l_i^5}$, the anti-spike inside can be annihilated by the spike, and the neuron returns to the initial state with no spike or anti-spike inside. Neuron σ_{l_j} fires at step $t + 10$, starting to simulates instruction l_j of M.
2. Neuron σ_r has six spikes (corresponding to the fact that the number stored in register r is 0) before it receives the anti-spike from neuron σ_{l_i}. By receiving the anti-spike from neuron $\sigma_{l_i^4}$, one spike in neuron σ_r is annihilated, which ends with five spikes. At step $t + 3$, neuron σ_r fires by using the rule $a^5 \rightarrow a$ and sends a spike to neurons $\sigma_{l_i^5}, \sigma_{l_i^6}$ and $\sigma_{l_i^7}$, respectively. The anti-spike in neuron $\sigma_{l_i^5}$ will be annihilated by the spike from neuron σ_r. So, neuron $\sigma_{l_i^5}$ ends with no anti-spike or spike inside and remains inactive. By receiving the spike from neuron σ_r, neurons $\sigma_{l_i^6}$ and $\sigma_{l_i^7}$ fire at step $t + 4$ sending two spikes to neuron $\sigma_{l_i^8}$. In neuron $\sigma_{l_i^8}$, rule $a^2 \rightarrow a$ is enabled, so it can fire at step $t + 5$ emitting one spike to neuron $\sigma_{l_i^{10}}$, which fires one step later sending a spike to neurons $\sigma_{l_i^{11}}, \sigma_{l_i^{12}}$ and $\sigma_{l_i^{13}}$, respectively. At step $t + 7$, the three neurons become active with emitting three spikes to neuron $\sigma_{l_i^{14}}$, as well as one spike to neuron $\sigma_{l_i^{15}}$. Having one spike inside, neuron $\sigma_{l_i^{15}}$ will fire and send a spike to neuron σ_{l_k} at step $t + 8$. Neuron σ_{l_k} can be activated one step later to start simulating instruction l_k of M.

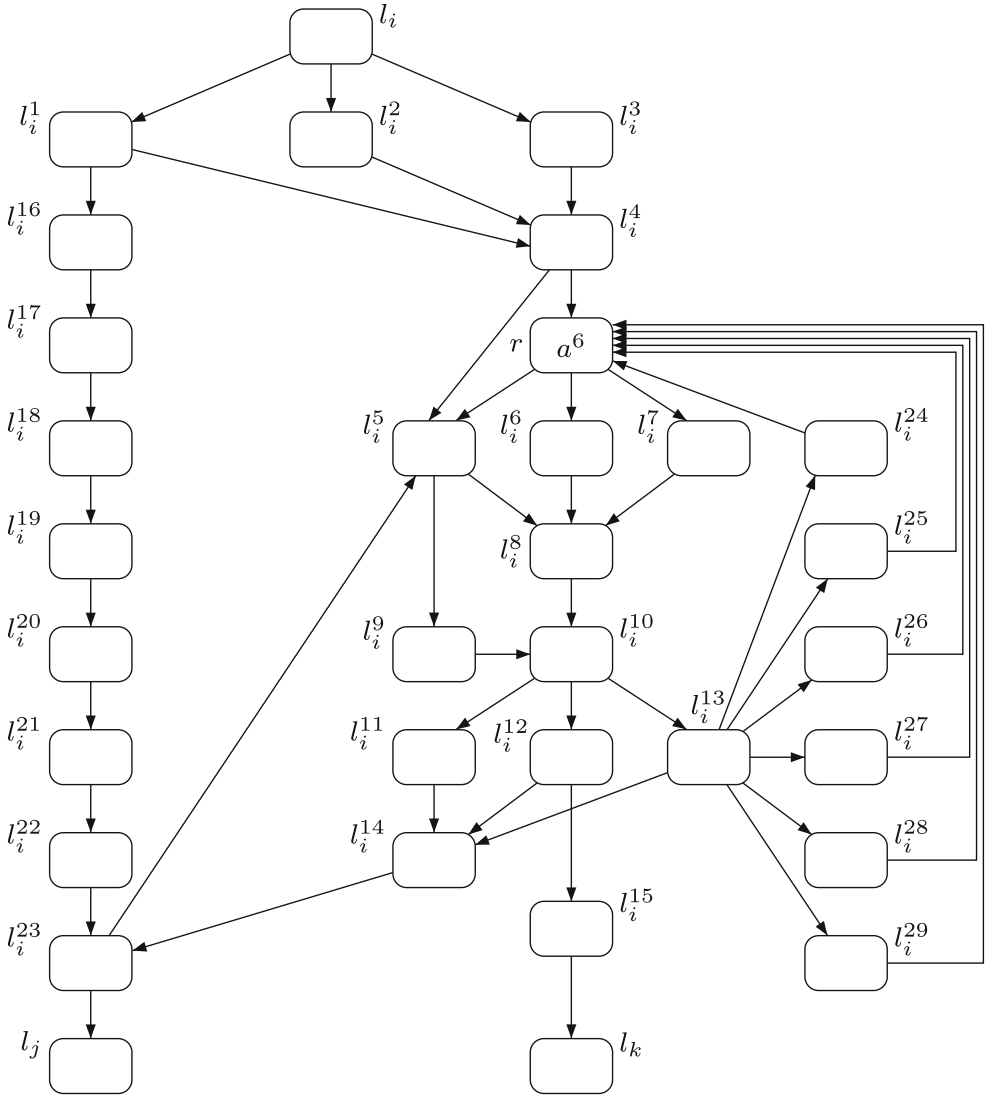

Fig. 3 Module SUB of Π (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$)

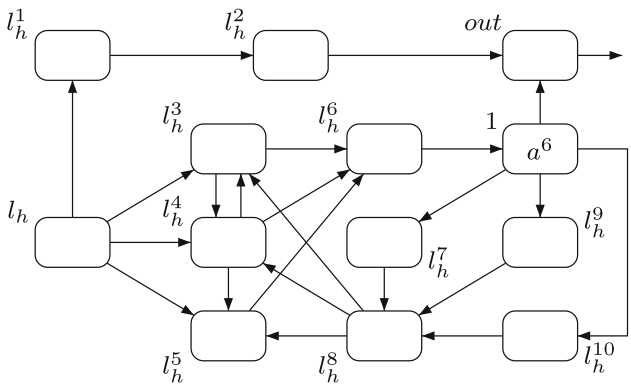

Fig. 4 Module FIN of Π (outputting the result of the computation)

Moreover, with three spikes inside, neuron $\sigma_{l_i^{14}}$ can produce an anti-spike and send it to neuron $\sigma_{l_i^{23}}$ at step $t + 8$. The anti-spike will annihilate the spike arriving in neuron $\sigma_{l_i^{23}}$ along path $\sigma_{l_i}, \sigma_{l_i^1}, \sigma_{l_i^{16}}, \ldots, \sigma_{l_i^{22}}$ at the same step, hence neuron $\sigma_{l_i^{23}}$ remains inactive in this case. Note that at step $t + 8$, six spikes are sent back to neuron σ_r from neurons $\sigma_{l_i^{24}}, \ldots \sigma_{l_i^{29}}$. In this way, when the system simulates instruction l_k of M, the number spikes in neuron σ_r is 2, which corresponds to the fact that the number stored in register r is 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 and decreased by one), or in neuron σ_{l_k} (if the number stored in register r is 0).

Note that there is no interference between the ADD modules and the SUB modules, other than correctly firing the neurons σ_{l_j} or σ_{l_k}, which may label instructions of the other kind. However, it is possible to have interference between two SUB modules. Specifically, if there are

several SUB instructions l_v that act on register r, then neuron σ_r has synapse connection to all neurons $\sigma_{l_v^5}, \sigma_{l_v^6}$ and $\sigma_{l_v^7}$. When a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with l_v ($l_v \neq l_i$) all neurons receive no spike except for neurons $\sigma_{l_v^5}, \sigma_{l_v^6}$ and $\sigma_{l_v^7}$. The three neurons will fire sending three spikes to neuron $\sigma_{l_v^8}$, as well as one spike to neuron $\sigma_{l_v^9}$. One step later, neuron $\sigma_{l_v^8}$ consumes the three spikes emitting an anti-spike to neuron $\sigma_{l_v^{10}}$, and neuron $\sigma_{l_v^9}$ fires by the rule $a \rightarrow a$ sending one spike to neuron $\sigma_{l_v^{10}}$. In neuron $\sigma_{l_v^{10}}$, the spike and anti-spike will annihilate each other. The above processes complete in three steps, and in this way, neurons $\sigma_{l_v^5}, \sigma_{l_v^6}$ and $\sigma_{l_v^7}$ return to the initial state that there is neither spike nor anti-spike inside. Consequently, the interference among SUB modules will not cause "undesired" steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M).

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

Assume now that the computation in M halts (that is, the halting instruction is reached), which means that neuron σ_{l_h} in Π has one spike and fires. At that moment, neuron σ_1 contains $n + 6$ spikes, for the number n stored in register 1 of M. We suppose it is at step t, neuron σ_{l_h} fires sending one spike to neurons $\sigma_{l_h^1}, \sigma_{l_h^3}, \sigma_{l_h^4}$ and $\sigma_{l_h^5}$. At step $t + 2$, one spike arrives in neuron σ_{out} along path $\sigma_{l_h}, \sigma_{l_h^1}, \sigma_{l_h^2}$ such that neuron σ_{out} fires for the first time at step $t + 3$ and emits the first spike to the environment. At step $t + 2$, neuron $\sigma_{l_h^6}$ fires sending an anti-spike to neuron σ_1, and from the moment on, neuron $\sigma_{l_h^6}$ continuously sends an anti-spike to neuron σ_1 in each step (because neurons $\sigma_{l_h^3}, \sigma_{l_h^4}$ and $\sigma_{l_h^5}$ continuously send three spikes to neuron $\sigma_{l_h^6}$ in each step). Neuron σ_1 remains inactive until step $t + n + 2$, when it has five spikes inside (since it receives $n + 1$ anti-spikes in total). It fires by using the rule $a^5 \rightarrow a$ and sends a spike to neuron σ_{out}. At step $t + n + 3$, neuron σ_{out} fires for the second time and sends the second spike to the environment. The distance of the two spikes from neuron σ_{out} is $(t + n + 3) - (t + 3) = n$, which is exactly the number stored in register 1.

When neuron σ_1 fires at step $t + n + 2$, it also sends one spike to neurons $\sigma_{l_h^7}, \sigma_{l_h^9}$ and $\sigma_{l_h^{10}}$, which will fire at step $t + n + 3$ sending three spikes to neuron $\sigma_{l_h^8}$. By using the rule $a^3 \rightarrow \bar{a}$, neuron $\sigma_{l_h^8}$ can fire in the next step and send an anti-spike to neurons $\sigma_{l_h^3}, \sigma_{l_h^4}, \sigma_{l_h^5}$ to stop them to emit spikes to neuron $\sigma_{l_h^6}$ (the spike in neurons $\sigma_{l_h^3}, \sigma_{l_h^4}, \sigma_{l_h^5}$ can be annihilated by the anti-spike), thus halting the whole system.

According to the function of the above three modules, it is clear that the register machine M can be correctly simulated by HASN P system Π. For the homogenous rule set of system Π, it contains two categories (identified by (a, a) and $(a, \bar{a})$) of pure form spiking rules. It is also worth to mention that there are only six rules in each neuron, and no forgetting rule is used. This concludes the proof. □

When HASN P systems working in the accepting mode, a specific input neuron σ_{in} is used to receive spikes from the environment. It is assumed that exactly two spikes are entering the system, and the number n of steps elapsed between the two spikes is the number checked to be accepted. After receiving the second spikes such system starts to work, and if it finally halts, then number n is said to be accepted.

Theorem 2 $N_{acc}HASNP(cate_2, prule_6) = NRE$ *with* $cate_2 = \{(a, a), (a, \bar{a})\}$.

Proof Let M be a deterministic register machine working in the accepting mode. An HASN P system Π working in the accepting mode 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, there are $n + 6$ spikes in neuron σ_r. In the simulation, the FIN module is not necessary, but an INPUT module is need to receive spike train $10^{n-1}1$ from the environment. After receiving the second spike in neuron σ_{in}, the system starts to work. The tasks of introducing two spikes to neuron σ_{in} and starting the system are covered by the INPUT module shown in Fig. 5. The system Π is also composed of the homogenous neurons as shown in Fig. 1.

Initially, all neurons in the INPUT module are empty with the exception of neurons σ_1 and σ_{c_5}, which contains six and seven spikes, respectively. By the INPUT module, number n is introduced in the system as a spike train $10^{n-1}1$ through neuron σ_{in}. Suppose that neuron σ_{in} fires

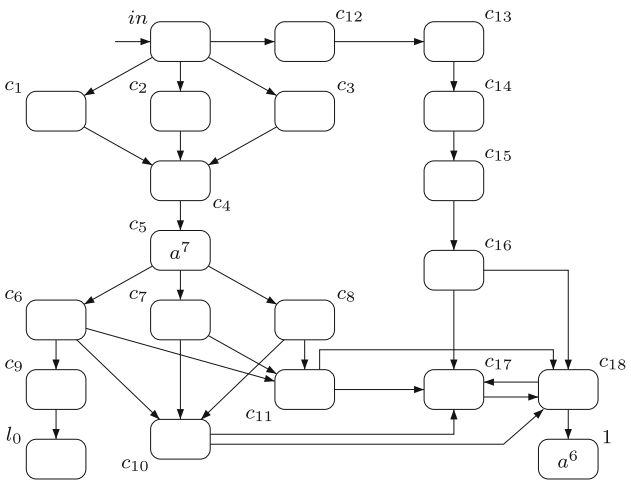

Fig. 5 The INPUT module of Π

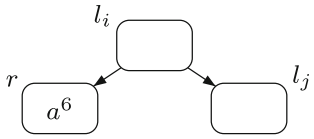

Fig. 6 The deterministic ADD module of Π

for the first time at step t by receiving the first spike from the environment. It sends one spike to neurons $\sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}$ and $\sigma_{c_{12}}$. At step $t + 2$, neuron σ_{c_4} fires sending an anti-spike to neuron σ_{c_5}. After doing the annihilation, neuron σ_{c_5} ends with six spikes and remains inactive. Moreover, at step $t + 5$, one spike arrives in neurons $\sigma_{c_{17}}$ and $\sigma_{c_{18}}$ by passing along path $\sigma_{c_{12}}, \sigma_{c_{13}}, \ldots, \sigma_{c_{16}}$. From step $t + 6$ onwards, neurons $\sigma_{c_{17}}$ and $\sigma_{c_{18}}$ begin to exchange a spike between them, and one spike is continuously emitted to neuron σ_1 from neuron $\sigma_{c_{18}}$ in each of the continuous steps.

At step $t + n$, the second spike enters neuron σ_{in}, so neuron fires σ_{in} for the second time at that moment. Two steps later, neuron σ_{c_4} fires for second time and sends an anti-spike to neuron σ_{c_5}. After annihilating that anti-spike, neuron σ_{c_5} ends with five spikes inside. At step $t + n + 3$, it fires and sends one spike to neurons $\sigma_{c_6}, \sigma_{c_7}$ and σ_{c_8}. At step $t + n + 5$, neurons $\sigma_{c_{10}}$ and $\sigma_{c_{11}}$ contain three spikes and fire by using rule $a^3 \to \bar{a}$. At that moment, two anti-spikes are sent to neurons $\sigma_{c_{17}}$ and $\sigma_{c_{18}}$. The two anti-spikes will make neuron $\sigma_{c_{18}}$ stop sending spikes to neuron σ_1. Neuron $\sigma_{c_{17}}$ contains no spike and anti-spike after annihilation, and it stops exchanging spikes with neuron $\sigma_{c_{16}}$.

From step $t + 6$ to $t + n + 5$, neuron σ_1 continuously receives one spike from neuron $\sigma_{c_{17}}$ in each step. It receives n spikes in total, that is, there are $n + 6$ spikes in it (six spikes are pre-stored in neuron σ_1). Number n can be introduced in the system in this way. At step $t + n + 5$, neuron σ_{l_0} receives one spike from neuron σ_{c_9} and becomes active by using the rule $a \to a$ at step $t + n + 6$.

Besides INPUT module, the deterministic ADD module is also needed to simulate the deterministic ADD instructions, which is indicted in Fig. 6. By receiving one spike, neuron σ_{l_i} can be activated and send one spike to neurons σ_r and σ_{l_j}. The number of spikes in neuron σ_r is increased by one, which simulates the number in register r increasing by one, while neuron σ_{l_j} can become active by using the rule $a \to a$ to start simulating instruction l_j of M.

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 by the spiking rule $a \to a$, but it sends no spike out because it has no outgoing synapses, and in this way, the work of system Π stops.

Initially, the number n to be computed is stored in neuron σ_1 in the form of $n + 6$ spikes. We can check that a computation of system Π stops if and only if the computation in M stops. The system is composed of neuron from Fig. 1, that is, each neuron contains two categories of pure form of spiking rules, and there are only six spiking rules in the neuron. □

4 Final remarks

In this work, inspired by the homogeneity of biological neurons (in biological neural networks, neurons are similar with each other), we consider a restricted variant of SN P systems with anti-spikes, called homogenous SN P systems with anti-spikes (shortly called HASN P systems). In the systems, each neuron has the same set of spiking and forgetting rules, that is, having the homogeneity. We investigate the computing power of HASN P systems. As a result, such systems without using forgetting rules can achieve the Turing completeness. These results provide a group of practically feasible computing models, since SN P systems with only pure form of rules can avoid the computationally hard problem hidden in checking whether a non-pure rule can be applied [8].

Many problems of HASN P systems need further researches. An interesting problem among them is to design a universal asynchronous homogenous SN P systems with anti-spikes. Also, it seems interesting to know how "small" such universal systems can become, with respect to the number or type of categories. Moreover, the language generated by such systems is worthy to be studied. From the applicable point of view, developing simulators for HASN P system is also deserved for further research. The ideas used in SN P systems may be also used in the research artificial neural networks, since the two kinds of computing models are both inspired from the biological neural systems.

References

1. Chen H, Freund R, Ionescu M, Păun G, Pérez-Jiménez MJ (2007) On string languages generated by spiking neural P systems. Fundamenta Informaticae 75(1–4):141–162
2. Francis GC, Henry A, Martínez-del-Amor MA (2011) An improved GPU simulator for spiking neural P systems. In: The proceedings of the 6th bio-inspired computing-theory and applications, Penang, Malaysia, pp 262–267
3. Francis GC, Henry A, Martínez-del-Amor MA (2011) A spiking neural P system simulator based on CUDA. In: The proceeding of international conference on membrane computing, pp 87–103
4. Ionescu M, Păun G, Yokomori T (2006) Spiking neural P systems. Fundamenta Informaticae 71(2–3):279–308

5. Ionescu M, Sburlan D (2007) Some applications of spiking neural P systems. The 8th workshop on membrane computing, Thessaloniki, pp 383–394
6. Ishdorj TO, Leporati A, Pan L, Zeng X, Zhang X (2010) Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Theoret Comput Sci 411(25):2345–2358
7. Jones R (2012) What males a human brain? Nat Rev Neurosci 13:655
8. Leporati A, Zandron C, Ferretti C, Mauri G (2007) On the computational power of spiking neural P systems. In: The proceedings of 5th brainstorming week on membrane computing, Seville: Fenix Editora, pp 227–246
9. Maass M, Bishop C (1999) Pulsed neural networks. MIT Press, Cambridge
10. Maass W (2002) Computing with spikes. Special Issue Found Inf Process TELEMATIK 8(1):32–36
11. Macías-Ramos LF, Pérez-Hurtado I., García-Quismondo M, Valencia-Cabrera L, Pérez-Jiménez MJ, Riscos-Núñez A (2012) A P-lingua based simulator for spiking neural P systems. Lect Notes Comput Sci 7184:257–281
12. Minsky M (1967) Computation—finite and infinite machines. Prentice Hall, New Jersey
13. Pan L, Păun G (2009) Spiking neural P systems with anti-spikes. Int J Comput Commun Control 4(3):273–282
14. Pan L, Păun G (2010) Spiking neural P systems: an improved normal form. Theoret Comput Sci 411(6):906–918
15. Pan L, Păun G, Pérez-Jiménez MJ (2011) Spiking neural P systems with neuron division and budding. Sci China Inf Sci 54(8):1596–1607
16. Pan L, Wang J, Hoogeboom HJ (2012) Spiking neural P systems with astrocytes. Neural Comput 24:1–24
17. Pan L, Zeng X (2011) Small universal spiking neural P systems working in exhaustive mode. IEEE Trans Nanobiosci 10(2):99–105
18. Pan L, Zeng X, Zhang X (2011) Time-free spiking neural P systems. Neural Comput 23(5):1320–1342
19. Pan L, Zeng X, Zhang X, Jiang Y (2012) Spiking neural P systems with weighted synapses. Neural Process Lett 35(1):13–27
20. Păun A, Păun G (2007) Small universal spiking neural P systems. BioSystems 90:48–60
21. Păun G (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143
22. Păun G (2009) A bibliography of spiking neural P system. In: The proceeding of 7th brainstorming week on membrane computing, sevilla, vol 2, pp 207–212
23. Păun G, Rozenberg G, Salomaa A (eds) (2010) Handbook of membrane computing. Oxford University Press, Oxford
24. Song T, Pan L, Păun G (2013) Asynchronous spiking neural P systems with local synchronization. Inf Sci 219:197–207
25. Thomas SP (2006) Evolution of the size and functional areas of the human brain. Ann Rev Anthropol 35:379–406
26. Wang J, Hoogeboom HJ, Pan L, Păun G, Pérez-Jiménez MJ (2010) Spiking neural P systems with weights. Neural Comput 22(10):2615–2646
27. 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