Universality of Spiking Neural P Systems with Anti-spikes

Venkata Padmavati Metta and Alica Kelemenová

Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic

Abstract. Spiking neural P systems with anti-spikes (in short, SN PA systems) are membrane systems that communicate using two types of objects called spikes and anti-spikes, inspired by neurons communicating through excitatory and inhibitory impulses. This paper shows that computational completeness in an SN PA systems can be achieved with neurons having only two pure spiking rules of the form $a \to a$ and $a \to \overline{a}$ without any forgetting rules. We also construct a small universal SN PA system with 91 simple neurons i.e., neurons having only one rule of the form $a \to \overline{a}$ or $a \to a$.

1 Introduction

Spiking neural P system [5] is a neural-inspired computational model based on the concept of spiking neurons. It consists of a set of neurons placed in the nodes of a directed graph (arcs representing synapses) and neurons communicate with each other using only one kind of objects called spikes, identical electrical impulses. The objects evolve by means of standard spiking rules, which are of the form $E/a^c \to a$, where E is a regular expression over $\{a\}$ and $c \geq 1$. The meaning is that a neuron containing k spikes such that $a^k \in L(E)$, $k \geq c$, can consume c spikes and produce one spike. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. There are also forgetting rules, of the form $a^s \to \lambda$ with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes. One neuron is distinguished as the output neuron and its spikes also exit into the environment, thus producing a binary sequence called spike train (moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0). The distance between consecutive spikes is the main way to encode information.

Spiking neural P system with anti-spikes [4] works in the same way as standard SN P system but deals with two types of objects called spikes (a) and anti-spikes ($\overline{a}$). The spiking rules are of the form $E/b^c \to b'$, where $b, b' \in \{a, \overline{a}\}$. If $L(E) = \{b^c\}$ then the rules are written as $b^c \to b'$ and are called pure. The system has four categories of spiking rules identified by (a, a), $(a, \overline{a})$ (anti-spikes are produced from usual spikes by means of usual spiking rules), $(\overline{a}, a)$ and $(\overline{a}, \overline{a})$ (rules consuming anti-spikes can produce spikes or anti-spikes). The latter two rules are generally avoided as they are quite unnatural. Each neuron in the system has an implicit

T V Gopal et al. (Eds.): TAMC 2014, LNCS 8402, pp. 352–365, 2014.

annihilation rule of the form $a\overline{a} \rightarrow \lambda$ (if an anti-spike meets a spike in a given neuron, then they annihilate each other (the disappearance of one a and one $\overline{a}$ takes no time), and this happens instantaneously in a maximal way.

The problem which "ingredients" are needed to achieve computational completeness or universality has been a challenging question for these kind of systems also. Several answers have been given, for instance in [4], it was proved that SN PA systems with pure spiking rules of categories (a, a), $(a, \overline{a})$, and $(\overline{a}, a)$ with forgetting rules are universal as number generators. Recently, Song et al. [7] proved that pure spiking rules of categories (a, a) and $(a, \overline{a})$ without forgetting rules , or spiking rules of categories $(\overline{a}, a)$ and $(a, \overline{a})$ without forgetting rules (the neurons change spikes to anti-spikes or change anti-spikes to spikes) are sufficient for universality as number generators. Zeng et al. [9] proved that homogeneous SN PA systems, i.e., SN PA systems where the rules in every neuron are identical, are universal.

All these systems consider spikes to represent the number and the number of spikes present in the neuron corresponding to a register as a function of the number stored in the register. In this paper, we make use of anti-spikes to represent the number stored in the register and number of anti-spikes in the neuron is equal to the number stored in the corresponding register. This avoids the use of rules of the form $\overline{a} \rightarrow a$ in the neuron to check its contents for zero. Since all neurons corresponding to registers are having the same rule $a \rightarrow a$, and no initial spikes/anti-spikes are present in any neurons corresponding to the registers, the output registers can be the subject of SUB instructions also.

This paper proves that only two rules of the form $a \rightarrow a$ and $a \rightarrow \overline{a}$ without any forgetting rules are sufficient for the universality of SN PA systems. It is also a natural and well investigated topic in computer science to look for small universal computing devices of various types. This topic was also considered for SN PA systems. In [8], a universal SN PA system with 75 neurons is constructed as a device of computing functions in which 125 rules, 6 types of neurons and 8 types of rules are used. In this work, the problem of constructing universal SN PA systems with a small number of rules is also investigated. Specifically, a universal SN P system with 91 simple neurons ("simple" in the sense that each neuron has only one rule, so a total of 91 rules) having the rules of the form $a \rightarrow a$ or $a \rightarrow \overline{a}$ is constructed for computing functions.

This paper is organized as follows. We start with Section 2 by giving a brief introduction about the SN P system with anti-spikes. In Section 3, we prove the computational completeness of SN PA systems with neurons having only two rules of the form $a \rightarrow a$ and $a \rightarrow \overline{a}$. Universal SN PA system is constructed in Section 4.

2 Prerequisites

We assume the reader to be familiar with formal languages and automata theory and spiking neural P systems. The reader can find details about them in [2], [1] etc.

For an alphabet V, V^* is the free monoid generated by V with respect to the concatenation operation and the identity λ (the empty string); the set of all non-empty strings over V, that is, $V^* - \{\lambda\}$, is denoted by V^+. The family of Turing computable sets of natural numbers is denoted by NRE (it is the family of length sets of recursively enumerable languages) and the family of Turing computable sets of vectors of natural numbers is denoted by $PsRE$.

We directly introduce the type of SN PA systems we investigate in this paper. (*SN P system with anti-spikes*) A spiking neural P system with anti-spikes, of degree $m \geq 1$, is a construct

$$\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m, syn, out), \text{ where}$$

1. $O = \{a, \overline{a}\}$ is a binary alphabet. a is called *spike* and $\overline{a}$ is called an *anti-spike*.
2. $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m$ are neurons, of the form

$$\sigma_i = (n_i, R_i), \ 1 \leq i \leq m, \text{ where}$$

   (a) $n_i \in \{0, 1, 2, \ldots\}$ is the initial number of spikes in the neuron σ_i;
   (b) R_i is a finite set of *rules* of the following two forms:
   (*i*) $E/b^r \to b'$ where $b, b' \in \{a, \overline{a}\}$, $r \geq 1$ and E is either a regular expression over a or $\overline{a}$;
   (*ii*) $b^s \to \lambda$ for some $s \geq 1$, with the restriction that $b^s \notin L(E)$ for any rule $E/b^r \to b'$ of type (*i*) from R_i;

   There are four categories of spiking rules identified by $(b, b') \in \{(a, a), (a, \overline{a}), (\overline{a}, a), (\overline{a}, \overline{a})\}$. Here, we allow rules of category $(b, b') \in \{(a, a), (a, \overline{a})\}$ but not the other two types.
3. $syn \subseteq \{1, 2, 3, \ldots, m\} \times \{1, 2, 3, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among cells);
4. $out \in \{1, 2, 3, \ldots, m\}$ indicates the output neuron.

A rule $E/b^r \to b'$ is applied as follows. If the neuron σ_i contains c spikes/anti-spikes, and $b^c \in L(E)$, $c \geq r$, then the rule can *fire*, and upon application, r spikes/anti-spikes are consumed (thus only $c - r$ remain in σ_i) and a spike/anti-spike is released, which will immediately exit the neuron. The spike/anti-spike emitted by neuron σ_i will pass immediately to all neurons σ_j such that $(i, j) \in syn$. That means transmission of spike/anti-spike takes no waiting time (since the rules do not specify a time delay), the spike/anti-spike will be available in neuron σ_j in the next step. There is an additional restriction that a and $\overline{a}$ cannot stay together, they annihilate each other. If a neuron has either objects a or objects $\overline{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^q and $\overline{a}^s$ inside, then immediately an annihilation rule $a\overline{a} \to \lambda$ (which is implicit in each neuron), is applied in a maximal manner, so that either a^{q-s} or $(\overline{a})^{s-q}$ remain for the next step, provided that $q \geq s$ or $s \geq q$, respectively. This mutual annihilation of spikes and anti-spikes takes no waiting time and the annihilation rule has priority over spiking and forgetting rules, so each neuron always contains either only spikes or anti-spikes. If we have a rule $E/b^r \to b'$ with $L(E) = \{b^r\}$, then we write it in the simplified form as $b^r \to b'$

and called pure. The rules of the form $b^s \to \lambda$, are forgetting rules. If neuron contains exactly s spikes/anti-spikes, then forgetting rule $b^s \to \lambda$ can be applied removing s spikes/anti-spikes from the neuron immediately.

The *configuration* of the system is described by $\mathcal{C} = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$, where β_i is the number of spikes/anti-spikes present in neuron σ_i. At any moment, if $\beta_i > 0$, it means that there are β_i spikes in neuron σ_i; if $\beta_i < 0$, it indicates that neuron σ_i contains β_i anti-spikes. The initial configuration is $\mathcal{C}_0 = \langle n_1, n_2, \ldots, n_m \rangle$.

A global clock is assumed and in each time unit, each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential locally: only (at most) one rule is used in each neuron. For example, if a neuron σ_i has two firing rules, $E_1/b^r \to b'$ and $E_2/b^c \to b'$ with $L(E_1) \cap L(E_2) \neq \emptyset$, then it is possible that each of the two rules can be applied, and in that case only one of them is chosen non-deterministically. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. In each step, all neurons which can use a rule of any type, spiking or forgetting, have to evolve, using a rule.

Using the rules in this way, we pass from one configuration of the system to another configuration; such a step is called a transition. For two configurations $\mathcal{C}$ and $\mathcal{C}'$ of Π we denote by $\mathcal{C} \Longrightarrow \mathcal{C}'$, if there is a direct transition from $\mathcal{C}$ to $\mathcal{C}'$ in Π.

A computation of Π is a finite or infinite sequence of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. A computation halts if it reaches a configuration where no rule can be used. SN PA systems can be used as computing devices in various ways. Here we will use them as generators of numbers. When using an SN PA system in the generative mode, we start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron. The output generated is 0 if no spikes exit the output neuron and the computation halts. The computations and the result of computations are defined in the same way as for usual SN P systems - but we consider the restriction that the output neuron produces only spikes, not also anti-spikes. We denote by $N_2(\Pi)$ the set of numbers computed by Π in this way.

We generalize the SN PA by allowing it to produce k outputs. A k-output SN PA Π has k output neurons, $O_1, \ldots, O_k$. We say that Π generates a k-tuple $(l_1, \ldots, l_k) \in N^k$ if, starting from the initial configuration, there is a sequence of steps such that each output neuron O_i generates exactly two spikes $a\ a$ (the times the pair $a\ a$ are generated may be different for different output neurons) and the time interval between the first a and the second a is l_i. Moreover, after all the output neurons have generated their pair of spikes, the system eventually halts, in the following sense: Π halts if it reaches a configuration where no neurons are fireable. The set of all k-tuples generated is denoted by $Ps_2(\Pi)$. We denote by $N_2S_aNP(cate_y, prule_k, cons_q)$ $[Ps_2S_aNP(cate_y, prule_k, cons_q)]$, the families of all sets $N_2(\Pi)$ $[Ps_2(\Pi), resp.]$ generated by SN PA systems with at most y categories of spiking rules, at most $k \geq 1$ pure rules (only spiking) in each neuron, with all spiking rules $b^r \to b'$ having $r \leq q$.

In order to compute a function $f : N^k \to N$, k natural numbers $n_1, \ldots, n_k$ are introduced into the system by "reading" from the environment a binary sequence $z = 10^{n_1-1}10^{n_2-1} \ldots 10^{n_k-1}1$. This means that the input neuron of Π receives a spike at each step corresponding to a digit 1 from string z and no spike otherwise. Note that $k+1$ spikes are exactly inputted; that is, it is assumed that no further spike is coming to the input neuron after the last spike. The result of the computation is encoded in the time distance between the first two spikes emitted by the system with the restriction that the system outputs exactly two spikes and halts (immediately after the second spike), hence it produces a spike train of the form $0^b10^{r-1}1$, for some $b \geq 0$ and with $r = f(n_1, \ldots, n_k)$.

3 Computational Completeness of SN PA Systems

We pass now to prove that SN PA systems with neurons having two rules of the form $a \to a$ and $a \to \overline{a}$ are universal as number generators.

In the following proof we use the characterization of NRE by means of register machines [6]. Such a device - in the non-deterministic version - 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 (labeling an ADD instruction), l_h is the halt label (assigned to instruction $HALT$), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. When it is useful, a label can be seen as a state of the machine, l_0 being the initial state, l_h the final/accepting state.

The labeled instructions are of the following forms:

1. $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j,l_k non-deterministically chosen),
2. $l_i : (SUB(r), l_j, l_k)$ (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),
3. $l_h : HALT$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number n present in register 0 (we assume that the registers are always numbered from 0 to $m-1$) at that time is said to be generated by M. It is known (see, [6]) that register machines generate all sets of numbers which are Turing computable.

It is also possible to consider register machines producing sets of vectors of natural numbers. In this case a distinguished set of k-registers (for some $k \geq 1$) is designated as the output registers. A k-tuple $(l_1, l_2, \ldots, l_k) \in N^k$ is generated if M eventually halts and the contents of the output registers are $l_1, l_2, \ldots, l_k$ respectively. Without loss of generality we may assume that in the halting configuration all the registers, except the output ones, are empty.

We will refer to a register machine with k-output registers (the other registers are auxiliary registers) as a k-output register machine. It is well known that a set S of k-tuples of numbers is generated by a k-output register machine if and only if S is recursively enumerable. Therefore they characterize $PsRE$.

Theorem 1. *Generating spiking neural P systems with anti-spikes with only two types of rules of the form $a \to a$ and $a \to \overline{a}$ are computationally complete, i.e., $N_2S_aNP(cate_2, prule_2, cons_1) = NRE$.*

Proof. Let $M = (m, H, l_0, l_h, I)$ be a register machine, having the properties specified above; the result of a computation is the number from register 0 and this register can be decremented during the computation.

What we want to do is to have SN PA Π constructed in such a way (1) to simulate the register machine M, and (2) to have its output neuron spiking only twice, at an interval of time which corresponds to a number computed by M.

Instead of specifying all technical details of the construction, we present the three main types of modules of the system Π, with the neurons, their rules, and their synapses represented graphically. In turn, simulating M means to simulate the ADD instructions and the SUB instructions. Thus, we will have a type of modules associated with ADD instructions, one associated with SUB instructions, and one dealing with the spiking of the output neuron (a FIN module). The modules of the three types are given in Figs. 1, 2 and 3 respectively.

For each register r of M, we consider a neuron σ_r in Π whose contents correspond to the contents of the register. Specifically, if the register r holds the number $n > 0$, then the neuron σ_r will contain n anti-spikes.

With each label l_i of an instruction in M, we also associate a neuron σ_{l_i} and some auxiliary neurons $\sigma_{l_{i_q}}$, $q = 1, 2, 3, \ldots$, thus precisely identified by label l_i. Initially, all these neurons are empty, with the exception of the neuron σ_{l_0} associated with the start label of M, which contains a single spike. This means that this neuron is activated. During the computation, the neuron σ_l which receives a spike will become active. Thus, simulating an instruction $l_i : (OP(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by OP, then introducing a spike in one of the neurons σ_{l_j}, σ_{l_k} which becomes in this way active. When activating the neuron σ_{l_h}, associated with the halting label of M, the computation in M is completely simulated in Π; we will then send to the environment two spikes with time gap between them equal to the number stored in the first register of M.

Simulating $l_i : (ADD(r), l_j, l_k)$ (module ADD in Fig. 1).

The initial instruction, that labeled with l_0, is an ADD instruction. Assume that we are in a step t when we have to simulate an instruction $l_i : (ADD(r), l_j, l_k)$, with a spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and even if some spikes are present in the auxiliary neurons and labels of the previous instruction executed, they will be cleared in the first step when σ_{l_i} fires, so simulating the ADD instruction correctly. Having a spike inside, neuron σ_{l_i} fires producing a spike. This spike will simultaneously go to neurons $\sigma_{l_{i_1}}$, $\sigma_{l_{i_2}}$, $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$. These four neurons fire at the step $t + 1$ with neuron $\sigma_{l_{i_3}}$ non-deterministically choosing any of its rules $a \to a$ or $a \to \overline{a}$. These rules

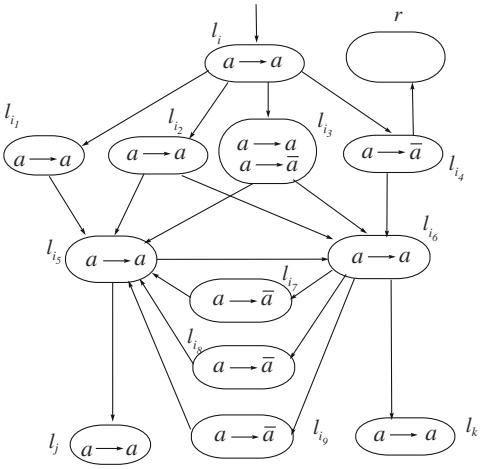

Fig. 1. ADD module: simulation of $l_i : (ADD(r), l_j, l_k)$

determine the non-deterministic choice of the neurons σ_{l_j} or σ_{l_k} to activate. If $a \to a$ is used in $\sigma_{l_{i_3}}$, then $\sigma_{l_{i_5}}$ receives three spikes, $\sigma_{l_{i_6}}$ receives a spike and $\sigma_{l_{i_4}}$ sends an anti-spike to σ_r (thus simulating the increase of the value of register r with 1), $\sigma_{l_{i_6}}$ uses its rule $a \to a$ and sends a spike to $\sigma_{l_{i_7}}$, $\sigma_{l_{i_8}}$, $\sigma_{l_{i_9}}$ and σ_{l_k}. At the step $t+3$, neurons $\sigma_{l_{i_7}}$, $\sigma_{l_{i_8}}$ and $\sigma_{l_{i_9}}$ fire using their rules $a \to \overline{a}$ and send three anti-spikes to $\sigma_{l_{i_5}}$ (here three spikes and three anti-spikes get annihilated). At the same step σ_{l_k} also becomes active, starting the simulation of the instruction l_k.

If $\sigma_{l_{i_3}}$ uses the rule $a \to \overline{a}$ at $t+1$, then the anti-spike from $\sigma_{l_{i_3}}$ and spike from $\sigma_{l_{i_2}}$ gets annihilated in both $\sigma_{l_{i_5}}$ and $\sigma_{l_{i_6}}$. Thus at step $t+2$, $\sigma_{l_{i_5}}$ has one spike and $\sigma_{l_{i_6}}$ has one anti-spike. Neuron $\sigma_{l_{i_5}}$ fires using its rule $a \to a$ sending a spike to $\sigma_{l_{i_6}}$ and σ_{l_j}. In $\sigma_{l_{i_6}}$, the spike gets annihilated with the anti-spike. At time $t+3$, neuron σ_{l_j} becomes active, thus starting the simulation of the instruction l_j.

Therefore, from the firing of neuron σ_{l_i}, the system adds one anti-spike to neuron σ_r and non-deterministically fires one of neurons σ_{l_j} and σ_{l_k}. Consequently, the simulation of the ADD instruction is possible in Π.

Simulating $l_i : (SUB(r), l_j, l_k)$ (module SUB in Fig. 2).

Assume that we are in a step t when we have to simulate an instruction $l_i : (SUB(r), l_j, l_k)$, with a spike present in neuron σ_{l_i}. Even though some spikes are present in the auxiliary neurons and labels of the previous instruction executed, they will be cleared in the first step when σ_{l_i} fires, so simulating the SUB instruction correctly. Let us examine now Fig. 2, starting from the situation of having a spike in neuron l_i and neuron σ_r, which holds a number of anti-spikes (this number is the value of the corresponding register r). The spike of neuron l_i goes immediately to two neurons, $\sigma_{l_{i_1}}$ and σ_r. If σ_r contains any anti-spikes (this corresponds to the case when register r is non-empty), then the spike gets annihilated with one anti-spike in σ_r, which means the contents of register r is

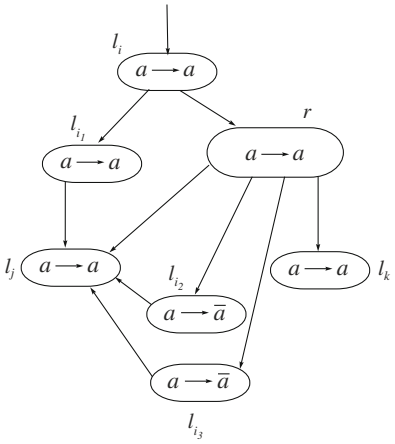

Fig. 2. SUB module: simulation of $l_i : (SUB(r), l_j, l_k)$

decremented by one. In step $t+1$ no spike will come out of σ_r while $\sigma_{l_{i_1}}$ fires and sends a spike to σ_{l_j} and thus activates the neuron σ_{l_j}. In step $t+2$, neuron σ_{l_j} fires, as requested by simulating the SUB instruction.

If in neuron σ_r there is no anti-spike (this corresponds to the case when register r is empty), then the rule $a \to a$ is used in σ_r at step $t+1$, hence the neuron σ_{l_j} receives two spikes and at the same time neurons $\sigma_{l_{i_2}}$ and $\sigma_{l_{i_3}}$ receive a spike. In the step $t+2$, neurons $\sigma_{l_{i_2}}$ and $\sigma_{l_{i_3}}$ fire and send two anti-spikes to σ_{l_j} and they get annihilated with the two spikes already present in the neuron σ_{l_j}. In the same step σ_{l_k} fires, This means that the simulation of the SUB instruction is correct, we started from l_i and we ended in l_j if the register was non-empty and decreased by one, and in l_k if the register was empty.

Simulating $l_h : (HALT)$ (module FIN in Fig. 3).

Assume now that the computation in M halts, which means that the halting instruction is reached. For Π this means that the neuron l_h gets a spike and fires. Let t be the moment when neuron l_h fires. At that moment, neuron σ_0 contains n anti-spikes, for n being the contents of register 0 of M. The spike of neuron l_h reaches immediately to neurons σ_0, $\sigma_{l_{h_1}}$, $\sigma_{l_{h_2}}$ and $\sigma_{l_{h_3}}$. It is important to remember that this neuron can be involved in a SUB instruction because we have the same rule $a \to a$ in each neuron that corresponds to any register in M.

If σ_0 has no anti-spikes (when the value in register 0 is 0), at moment $t+1$, four neurons $\sigma_{l_{h_1}}$, $\sigma_{l_{h_2}}$, $\sigma_{l_{h_3}}$ and σ_0 fire and all of them spike immediately. Neuron $\sigma_{l_{h_1}}$ sends a spike to $\sigma_{l_{h_6}}$, neuron $\sigma_{l_{h_2}}$ sends a spike to σ_0, neuron σ_0 sends its spike to $\sigma_{l_{h_4}}$ and $\sigma_{l_{h_5}}$, while $\sigma_{l_{h_2}}$ and $\sigma_{l_{h_3}}$ exchange their spikes. At the step $t+2$, neurons $\sigma_{l_{h_4}}$, $\sigma_{l_{h_5}}$, $\sigma_{l_{h_6}}$ and σ_0 fire whereas neurons $\sigma_{l_{h_2}}$ and $\sigma_{l_{h_3}}$ will not fire since they have two spikes in each. The spike from neuron σ_0 and the anti-spike from neuron $\sigma_{l_{h_4}}$ are annihilated in $\sigma_{l_{h_5}}$. Neurons σ_{out} and $\sigma_{l_{h_4}}$ receive two spikes each, so cannot fire in the next step and the system halts without sending any spikes to the environment, denoting that the number 0 is generated by the system.

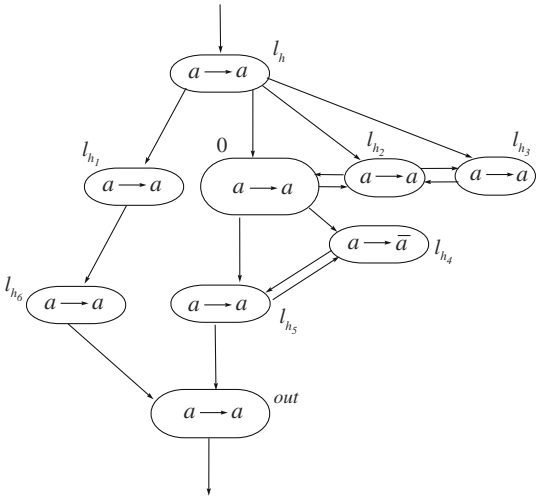

Fig. 3. FIN module: simulation of $l_i : HALT$

If σ_0 has $n > 0$ anti-spikes (when the value of register 0 is $n > 0$), we can observe from the Fig. 3 that at the time $t + 1$, only three neurons $\sigma_{l_{h_1}}$,$\sigma_{l_{h_2}}$, $\sigma_{l_{h_3}}$ (neuron σ_0 will not fire as the incoming spike is annihilated with one of its anti-spikes). Neuron $\sigma_{l_{h_1}}$ sends a spike to $\sigma_{l_{h_6}}$, neuron $\sigma_{l_{h_2}}$ sends a spike to σ_0, while $\sigma_{l_{h_2}}$ and $\sigma_{l_{h_3}}$ exchange their spikes. At the step $t + 2$, neurons $\sigma_{l_{h_2}}$, $\sigma_{l_{h_3}}$, $\sigma_{l_{h_5}}$ and $\sigma_{l_{h_6}}$ fire. The spike from neuron $\sigma_{l_{h_6}}$ is sent to neuron σ_{out}. So the neuron σ_{out} first fires in step $t + 3$ and sends its spike to the environment. The number of steps from this spike to the next one is the number computed by the system. In each step from $t + 1$ onwards neurons $\sigma_{l_{h_2}}$ and $\sigma_{l_{h_3}}$ exchange their spikes and $\sigma_{l_{h_2}}$ sends one spike to σ_0. The neuron σ_0 does not fire until it has any anti-spikes. This means that the process of removing anti-spikes from neuron σ_0 continues, iteratively having neuron $\sigma_{l_{h_2}}$ sending spikes until σ_0 has no anti-spikes. Thus the neuron σ_0 fires at the step $t + n + 1$ for the first time (for n being the initial number of anti-spikes of neuron σ_0 at time t). Neuron σ_0 sends a spike to $\sigma_{l_{h_2}}$, $\sigma_{l_{h_4}}$ and $\sigma_{l_{h_5}}$. The remaining steps work in the same way as in the previous case. At the step $t+n+3$ neurons σ_{out} spikes and the system halts.

The interval between the two spikes of neuron σ_{out} is $(t+n+3)-(t+3) = n$, exactly the value of register 0 of M in the moment when its computation halts. Consequently, $N_2(\Pi) = N(M)$ and this completes the proof. □

This result can have a nice interpretation: it is sufficient for a "brain" (in the form of an SN P system with anti-spikes) to have neurons sending either excitatory or inhibitory impulses which behaves non-deterministically in order to achieve "complete (Turing) creativity".

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_h), l_{19} : (SUB(0), l_0, l_{18}), l_{20} : (ADD(0), l_0),

l_{21} : (ADD(4), l_{18}), l_h : HALT

Fig. 4. A universal register machine M_u from Korec [3]

Theorem 1 can be easily extended by allowing more output neurons and then simulating a k-output register machine, producing in this way sets of vectors of natural numbers.

Theorem 2. $Ps_2S_aNP(cate_2, prule_2, cons_1) = PsRE$.

4 A Small Universal SN P System with Anti-spikes

A register machine specified above can also compute any Turing computable function: we introduce the arguments $n_1, n_2, \ldots, n_k$ in specified registers r_1, $r_2, \ldots, r_k$ (without loss of the generality, we may assume that we use the first k registers), we start with the instruction with label l_0, and if we stop (with the instruction with label l_h), then the value of the function is placed in another specified register, r_t, with all registers different from r_t being empty. The partial function computed in this way is denoted by $M(n_1, n_2, \ldots, n_k)$. In the computing form, the register machines can be considered deterministic, without losing the Turing completeness: all ADD instructions $l_i : (ADD(r), l_j, l_k)$ have $l_j = l_k$ (and the instruction is written in the form $l_i : (ADD(r), l_j)$).

In [3], the register machines are used for computing functions, with the universality defined as follows. Let $(\phi_0, \phi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y we have $\phi_x(y) = M_u(g(x), y)$. In [3], several universal register machines are constructed, with the input (the couple of numbers $g(x)$ and y) introduced in registers 1 and 2, and the result obtained in register 0.

We give now in the notation introduced above the specific universal register machine from [3], which will be used in this section: we have $M_u = (8, H, l_0, l_h, I)$, with the instructions (their labels constitute the set H) presented in Fig. 4 (The

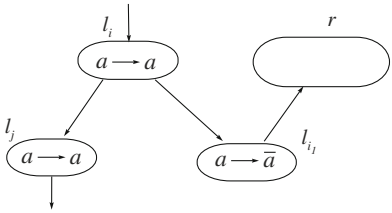

Fig. 5. Deterministic ADD module $l_i : (ADD(r), l_j)$

machine from [3] contains a separate check for zero of register 6, of the form l_8 : if register(6) = 0, then go to l_0, else go to l_{10}; this instruction was replaced in our set up by $l_8 : (SUB(6), l_9, l_0)$, $l_9 : (ADD(6), l_{10})$). Therefore, there are 8 registers (numbered from 0 to 7) and 23 instructions (hence 23 labels), the last instruction being the halting one. The input numbers (the "code" of the partial recursive function to simulate and the argument for this function) are introduced in registers 1 and 2, and the result is obtained in register 0.

In the systems constructed in this work, the neurons are quite "simple" in the sense that each neuron has only one rule.

We proceed now to constructing the universal SN PA system Π_u using pure rules of category (a, a) and $(a, \overline{a})$ without forgetting rules, for computing functions. To this aim, we follow a similar way used in previous section but to simulate a deterministic register machine by an SN PA system. Neurons are associated with each register and with each label of an instruction of the machine. If a register contains a number n, then the associated neuron will contain n anti-spikes. Modules as in Fig. 5 and Fig. 2 are associated with the ADD and the SUB instructions (each of these modules contains auxiliary neurons which do not correspond to registers or to labels of instructions).

The work of the system is triggered by introducing a spike in the neuron σ_{l_0} (associated with the starting instruction of the register machine). In general, the simulation of an ADD or SUB instruction starts by introducing a spike in the neuron with the instruction label (we say that this neuron is activated).

Starting with neurons σ_1 and σ_2 already loaded with $g(x)$ and y spikes, respectively, and introducing a spike in neuron σ_{l_0}, we can compute in our system Π_u in the same way as the universal register machine M_u from Fig. 4; if the computation halts, then neuron σ_0 will contain the $\phi_x(y)$ number of anti-spikes.

There are two additional tasks to solve: to introduce the mentioned anti-spikes in the neurons σ_1, σ_2, and to output the computed number. The first task is covered by module $INPUT$ presented in Fig. 6. The neuron σ_{c_5} converts the spikes it receives from the input neuron into anti-spikes. The neuron σ_{c_8} fires only after receiving the third anti-spike from σ_{c_5}, and then it sends a spike to neuron σ_{l_0}, thus starting the simulation of M_u. At that moment, neurons σ_1 and σ_2 are already loaded: neuron σ_{c_3} sends to neuron σ_1 as many anti-spikes as the number of steps between the first two input spikes, and after that it gets "over flooded" by the second input spike and is blocked (neurons σ_{c_1} and σ_{c_2} supply spikes to σ_{c_3} till they receive second spike through σ_{in}); in turn, neuron σ_{c_5}

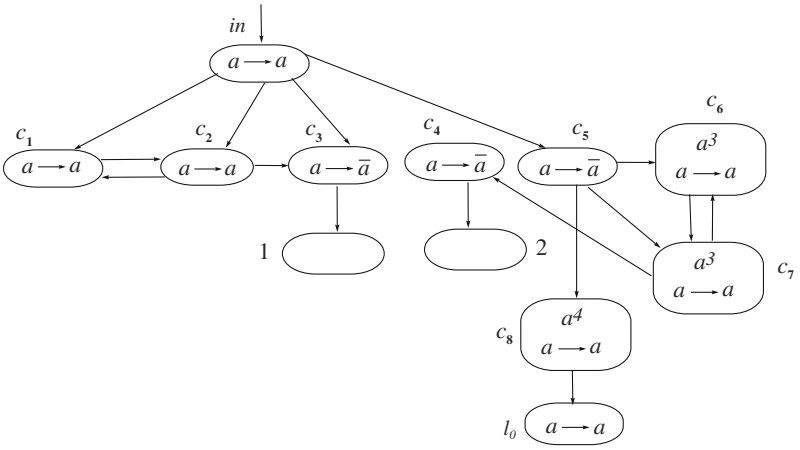

Fig. 6. $INPUT$ module

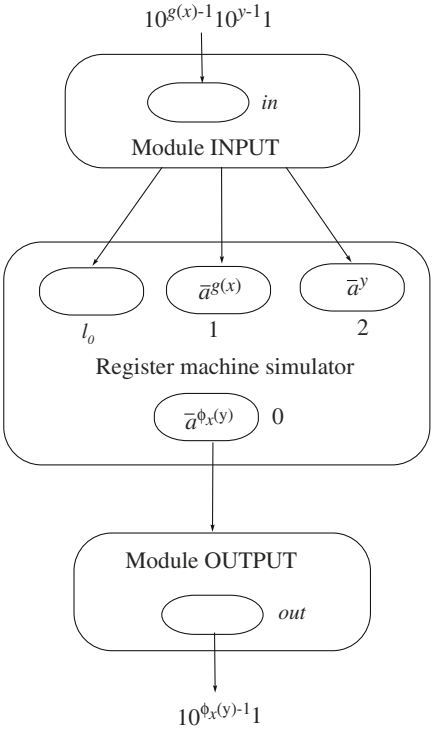

Fig. 7. The general design of the universal SN PA system

sends anti-spikes to neurons σ_{c_6}, σ_{c_7} and they start working only after collecting two anti-spikes. Neurons σ_{c_6} and σ_{c_7} supply one spike in each step to neuron σ_{c_4}, which loads σ_2 with as many anti-spikes as the number of steps between

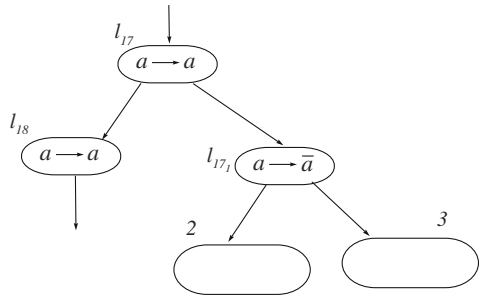

Fig. 8. A module simulating two consecutive ADD instructions

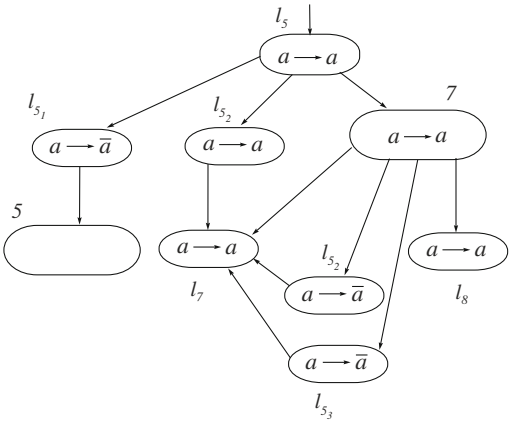

Fig. 9. A module simulating $ADD - SUB$ instructions

the last two input spikes and all three neurons stop working after receiving the third anti-spike from σ_{c_5}.

In this construction, we do not need to modify the universal register machine as in [8] for not allowing subtraction operations on the neuron where we place the result. So the result will be in the neuron σ_0 which corresponds to the register 0 of M_u.

Having the result of the computation in register 0, we can output the result by means of the module $OUTPUT$ which is same as FIN module in Fig. 3 (the working of this module is explained in the previous section). The overall design of the system is given in Fig. 7.

We can check that each neuron in the system Π_u has only one rule; that is, the system Π_u is simple. The system Π_u has 8 neurons for the 8 registers, 23 neurons for the 23 labels, 9 neurons for the 9 ADD instructions, 39 neurons for 13 SUB instructions, 9 neurons in the $INPUT$ module and 6 neurons in the $OUTPUT$ module which comes to a total of 94 neurons. This number can be slightly decreased, by some "code optimization", exploiting some particularities of the register machine M_u.

First, let us observe that the sequence of two consecutive ADD instructions: $l_{17} : (ADD(2), l_{21}), l_{21} : (ADD(3), l_{18})$, without any other instruction addressing the label l_{21}, can be simulated by the module from Fig. 8, and in this way we save a neuron associated with l_{21}.

The module from Fig. 9 can simulate the consecutive $ADD - SUB$ instructions $l_5 : (ADD(5), l_6)$, $l_6 : (SUB(7), l_7, l_8)$. A similar module can be constructed to simulate the consecutive ADD-SUB instructions $l_9 : (ADD(6), l_{10})$, $l_{10} : (SUB(4), l_0, l_{11})$. So two neurons (associated with the labels l_6 and l_{10}) are saved. We save a total of 3 neurons and get the improvement from 94 to 91 neurons. We state this result in the form of a theorem in order to stress its importance:

Theorem 3. *There exists a universal simple SN PA system with 91 neurons for computing functions.*

5 Conclusion

By using the characterization of types of rules, we were able to show that for obtaining computational completeness of spiking neural P system with anti-spikes only two rules of the form $a \to a$, $a \to \overline{a}$ are needed. In this work, the problem of constructing universal SN PA systems with a small number of rules is also investigated. The systems constructed in this work has 91 rules of the form $a \to a$ or $a \to \overline{a}$. It is possible to use less neurons to construct universal SN PA systems provided that neurons have more types of spiking rules.

References

1. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press (2010)
2. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, 3 volumes. Springer, Berlin (1998)
3. Korec, I.: Small universal Turing machines. Theoretical Computer Science 168, 267–301 (1996)
4. Pan, L., Păun, G.: Spiking Neural P Systems with Anti-Spikes. International Journal of Computers, Communications and Control 4(3), 273–282 (2009)
5. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae 71, 279–308 (2006)
6. Minsky, M.: Computation finite and infinite machines. Prentice Hall, Englewood Cliffs (1967)
7. Song, T., Pan, L., Wang, J., Venkat, I., Subramanian, K.G., Abdullah, R.: Normal Forms of Spiking Neural P systems with anti-spikes. IEEE Transactions on Nanobioscience 11(4), 352–359 (2012)
8. Song, T., Jiang, Y., Shi, X., Zeng, X.: Small Universal Spiking Neural P Systems with Anti-Spikes. Journal of Computational and Theoretical Nanoscience 10(4), 999–1006 (2013)
9. Zeng, X., Zhang, X., Pan, L.: Homogeneous spiking neural P systems. Fundamenta Informaticae 97(12), 1–20 (2009)