Smaller Universal Spiking Neural P Systems with Anti-Spikes

Venkata Padmavati Metta, Alica Kelemenová

Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
vmetta@gmail.com, alice.kelemenova@fpf.slu.cz

Abstract. Spiking neural P systems with anti-spikes (in short, SN PA systems) are parallel computing models based on the way neurons communicate using two types of electrical impulses called excitatory impulses (spikes) and inhibitory impulses (anti-spikes). A universal computing SN PA system was designed by T. Song et al. using 75 neurons with 6 different types of neurons, 8 different types of rules and a total of 125 rules. In this paper, we continue the study small universal spiking neural P systems with anti-spikes and we improve in the types of neurons and the number of rules. We construct a small universal SN PA system with 104 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 systems (in short, SN P systems) are membrane computing models which abstract the way neurons communicate by means of electrical impulses of identical shape, called spikes. The SN P systems were introduced in [6], and then investigated in a large number of papers. We refer to the respective chapter of [2] for general information in this area, and to the membrane computing website from [8] for details.

Spiking neural P systems with anti-spikes [5] consist of a set of neurons placed in the nodes of a directed graph and sending two types of signals (spikes, denoted by the symbol a and anti-spikes, denoted by the symbol $\overline{a}$) along synapses (arcs of the graph). Thus, the architecture is same as that of a spiking neural P system, but with two kinds of objects present in the neurons. The objects evolve by means of spiking rules, which 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 annihilation rule of the form $a\overline{a} \to \lambda$ (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 system works in a synchronized manner,

i.e., in each time unit, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the output neuron, and its spikes are also sent 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, it might be infinite if the computation does not stop. The distance between consecutive spikes is the main way to encode the information. SN PA systems with anti-spikes are proved as universal [5].

When investigating any (universal) computing device, a "standard" problem is to consider restricted versions of it without losing the computing power. This issue is also particularly interesting for SN PA systems. Several answers have been given, for instance in [5], 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. [9] 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. [11] 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 a number and the number of spikes present in a neuron (corresponding to a register) to represent the number stored in the register. In this paper, we make use of anti-spikes to represent a number and the number of anti-spikes in a neuron is equal to the number stored in the corresponding register. This avoids the use of rules of the form $\overline{a} \to a$ in a neuron to check its contents for zero. Here one extra neuron is maintained for each register which keeps a copy of the contents of the register but with a different spiking rule $a \to \overline{a}$, this reduces the number of auxiliary neurons required for the simulation of the SUB instruction.

It is 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. This paper considers only two rules of the form $a \to a$ and $a \to \overline{a}$ without any forgetting rules for constructing small universal SN PA systems. In [10], 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 less number of rules is investigated. Specifically, a universal SN P system with 104 simple neurons ("simple" in the sense that each neuron has only one rule, so a total of 104 rules) having the rules of the form $a \to a$ or $a \to \overline{a}$ is constructed for computing functions.

2 Prerequisites

We assume the reader to be familiar with formal language theory and membrane computing. The reader can find details about them in [3], [2] 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^+. When $V = \{a\}$ is a singleton, then we write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$.

A regular expression over an alphabet V is defined as: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (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 V.

We pass now to introducing the universal register machines, and then to the spiking neural P systems with anti-spikes.

2.1 Universal Register Machines

Because the register machines used in the following sections are deterministic, we only recall the definition of these types of machines. A deterministic 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 (labelling 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 labelled instructions are of the following forms:

1. $l_i : (ADD(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j),
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, [7]) that register machines generate all sets of numbers which are Turing computable.

A register machine 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, it is known that (see e.g., [7]) that the deterministic register machines are equivalent with Turing machines.

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(3), l_{18}), l_h: HALT

Fig. 1. A universal register machine M_u from Korec [4]

In [4], 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 [4], 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. In the following, we consider the specific universal register machine $M_u = (8, H, l_0, l_h, I)$, with instructions (their labels constitute the set H) given in Fig. 1, which is also the one used in [1] (it has 8 registers numbered from 0 to 7 and 23 instructions).

As in the case considered in [1], we further add a register 8, which is never decremented during the computation, and we replace the old halt instruction of M_u with the following three instructions: $l_h : (SUB(0), l_{22}, l'_h)$, $l_{22} : (ADD(8), l_h)$, and $l'_h : HALT$. Thus at the end of the program of the initial machine we will copy the contents of the old output register (register 0) into the register 8 which will serve as the new output register. We do this so that no subtraction rule will be applicable to the output register. In this way, we have 9 registers (numbered from 0 to 8), 24 ADD and SUB instructions, and 25 labels. We denote by M'_u the obtained register machine.

2.2 Spiking Neural P Systems with anti-spikes

We now introduce the SN P systems with anti-spikes in the form necessary for computing functions. A computing SN P system with anti-spikes, of degree

$m \geq 1$, is a construct of the form

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

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) $E/b^r \rightarrow b'$ where $b, b' \in \{a, \overline{a}\}$, $r \geq 1$ and E is either a regular expression over a or $\overline{a}$;
   (ii) $b^s \rightarrow \lambda$ for some $s \geq 1$, with the restriction that $b^s \notin L(E)$ for any rule $E/b^r \rightarrow b'$ 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 categories $(b, b') \in \{(a, a),\ (a, \overline{a})\}$ 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. $in,\ out \in \{1, 2, 3, \ldots, m\}$ indicate the input and the output neurons, respectively.

A rule $E/b^r \rightarrow b'$ 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 $a\overline{a} \rightarrow \lambda$ (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 $E/b^r \rightarrow b'$ with $L(E) = \{b^r\}$, then we write it in the simplified form as $b^r \rightarrow b'$ and called pure. The rules of the form $b^s \rightarrow \lambda$, are forgetting rules. If the neuron contains exactly s spikes/anti-spikes, then the forgetting rule $b^s \rightarrow \lambda$ 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.

Like in the case considered in [1], in order to compute a function $f : N^k \to N$, where N is the set of all non-negative integers, 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.

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 result 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. So 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 Small Universal SN P Systems with Anti-spikes

We proceed now to constructing the universal SN PA system Π_u using pure rules of categories (a, a) and $(a, \overline{a})$ without forgetting rules, for computing functions. The neurons in the system are quite "simple" in the sense that each neuron has only one rule. To construct a universal SN PA system Π_u, we follow the way used in [1] to simulate a deterministic register machine by an SN P system.

The usual way of simulating a register machine M'_u by an SN PA system consists in the construction of an SN P system with anti-spikes Π_u, where neurons are associated with each register and with each label of an instruction of the

machine. In this specific case, we associate a neuron σ_r for each register r and a copy of the contents of σ_r are kept in neuron σ_{b_r}. But σ_r has a spiking rule $a \to a$ where as σ_{b_r} has a spiking rule $a \to \overline{a}$. Keeping two copies of the contents of the register r with different spiking rules decreases the number of auxiliary neurons required in the simulation of a SUB instruction. If a register r contains a number n, then the associated neurons σ_r and σ_{b_r} will contain n anti-spikes each.

With each label l_i of an instruction in M'_u, 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. Specifically, modules ADD and SUB are constructed to simulate the instructions of M'_u, as well as $INPUT$ module which introduces a spike in neuron σ_{l_0} and needed anti-spikes in σ_1 and σ_2, and an $OUTPUT$ module which provides the computed number.

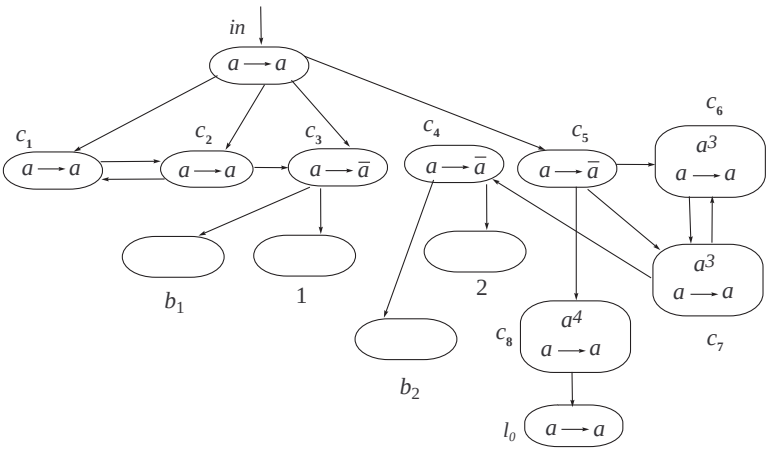

Fig. 2. $INPUT$ module

Instead of presenting these modules formally, we give them in the graphical form. In the initial configuration, all neurons of Π_u are empty. The task to introduce $g(x)$ and y anti-spikes in the neurons σ_1 and σ_2 respectively, is covered by the module $INPUT$ presented in Fig. 2. 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 neurons σ_1 and σ_{b_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} 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 and σ_{b_2} with as many anti-spikes as the number of steps between the last two input spikes and all three neurons stop working after receiving the third anti-spike from σ_{c_5}.

The work of the system is triggered by loading neurons σ_1 and σ_2 with $g(x)$ and y anti-spikes respectively, and introducing a spike in the neuron σ_{l_0} (associated with the starting instruction of the register machine). We can compute in our system Π_u in the same way as the universal register machine M'_u; if the computation halts, then neuron σ_8 will contain $\phi_x(y)$ number of anti-spikes.

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). Modules as in Fig. 3 and Fig. 4 are associated with the ADD and the SUB instructions.

Simulating $l_i : (ADD(r), l_j)$ (module ADD in Fig. 3).

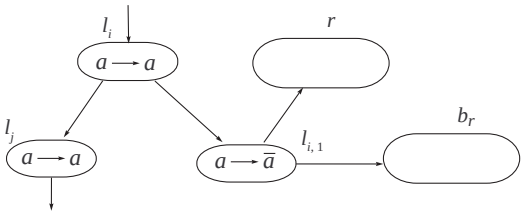

Fig. 3. Deterministic ADD module $l_i : (ADD(r), l_j)$

Assume that we are in a step t when we have to simulate an instruction $l_i : (ADD(r), l_j)$, with a spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spikes/anti-spikes in any other neurons, except in those associated with registers. Having a spike inside, neuron σ_{l_i} fires producing a spike. This spike will simultaneously go to neurons $\sigma_{l_{i,1}}$ and σ_{l_j}. In step $t+1$, neuron $\sigma_{l_{i,1}}$ fires using its rules $a \to \overline{a}$ and send an anti-spike to both σ_r and σ_{b_r}. Therefore, from the firing of neuron σ_{l_i}, the system adds one anti-spike each to neurons σ_r and σ_{b_r} and fires the neuron σ_{l_j}. Consequently, the simulation of the ADD instruction is possible in Π_u.

Simulating $l_i : (SUB(r), l_j, l_k)$ (module SUB in Fig. 4).

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 the neuron σ_{l_i} and no spikes/anti-spikes in any other neurons, except in those associated with registers. Let us examine now Fig. 4, starting from the situation of having a spike in neuron l_i and having the same number of anti-spikes in the neurons σ_r and σ_{b_r} (this number is the value of the corresponding register r). The spike of neuron l_i goes immediately to neurons $\sigma_{l_{i,1}}$, σ_r and σ_{b_r}. Now there are two possible cases.

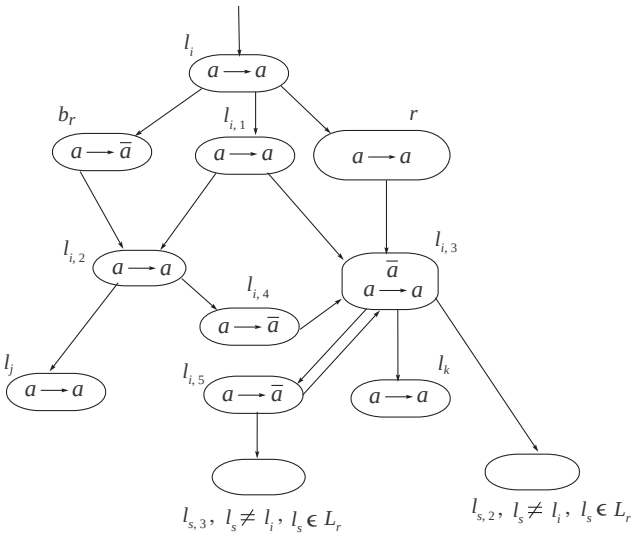

Fig. 4. SUB module: simulation of $l_i : (SUB(r), l_j, l_k)$, where $L_r = \{l \mid l$ is a label of a SUB instruction acting on the register $r\}$

1. At step t, if the neurons σ_r and σ_{b_r} have no anti-spikes (this corresponds to the case when register r is empty), then at step $t+1$, neurons $\sigma_{l_{i,1}}$, σ_r and σ_{b_r} fire, the rule $a \to a$ is used in σ_r and the rule $a \to \overline{a}$ is used in σ_{b_r}. Let L_r be the set of labels of SUB instructions in M'_u acting on register r. Neuron σ_r has synapses with all the neurons $\sigma_{l_{s,3}}$ and neuron σ_{b_r} has synapses with all neurons $\sigma_{l_{s,2}}$, where $l_s \in L_r$. After the step $t+1$, the neuron $\sigma_{l_{i,2}}$ has no spikes (since the spike from $\sigma_{l_{i,1}}$ gets annihilated with the anti-spike from σ_{b_r}) and the neuron $\sigma_{l_{i,3}}$ has a spike ($\sigma_{l_{i,3}}$ receives two spikes and one spike gets annihilated with the anti-spike initially present in it). In all the neurons $\sigma_{l_{s,3}}$, the spike coming from the neuron σ_r gets annihilated with the anti-spike present in them and all neurons $\sigma_{l_{s,2}}$ are left with an anti-spike, where $l_s \neq l_i$ and $l_s \in L_r$. Now at step $t+2$, neuron $\sigma_{l_{i,3}}$ fires sending a spike to σ_{l_k}, $\sigma_{l_{i,5}}$ and to all the neurons $\sigma_{l_{s,2}}$, $l_s \neq l_i$. So all the neurons $\sigma_{l_{s,2}}$ have no spikes/anti-spikes left inside. In the next step, the spike in $\sigma_{l_{l,5}}$ is converted into an anti-spike and sent to all $\sigma_{l_{s,3}}$, $l_s \in L_r$. In this way, the anti-spikes in neurons $\sigma_{l_{s,3}}$, $l_s \in L_r$ are reset to the value they have at the beginning of the simulation, which ensures that another SUB instruction can be simulated correctly at a subsequent step. When the neuron σ_{l_k} fires in the same step, system starts to simulate the instruction l_k of M'_u.
2. At step t, if the neuron σ_r contains any anti-spikes (this corresponds to the case when register r is non-empty), then the spike from σ_{l_i} gets annihilated with one of the anti-spikes in σ_r, which means the contents of register r is decremented by one. The same happens in the neuron σ_{b_r} also. In the next

step no spike will come out of σ_r and σ_{b_r} while $\sigma_{l_{i,1}}$ fires and sends a spike to $\sigma_{l_{i,2}}$ and $\sigma_{l_{i,3}}$. In $\sigma_{l_{i,3}}$, the spike gets annihilated with the anti-spike where as $\sigma_{l_{i,2}}$ is left with a spike. At step $t+2$, neuron $\sigma_{l_{i,2}}$ fires and sends a spike to neurons σ_{l_j} and $\sigma_{l_{i,4}}$. In the next step, the neuron σ_{l_j} starts and the neuron $\sigma_{l_{i,4}}$ restores the anti-spike in $\sigma_{l_{i,3}}$.

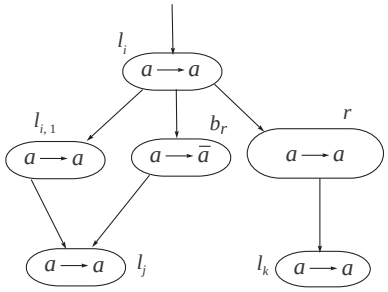

Fig. 5. SUB module: simulation of $l_i : (SUB(r), l_j, l_k)$, when L_r contains only one SUB instruction

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 is non-empty and is decreased by one, and in l_k if the register is empty. If L_r contains only one SUB instruction for a register r, then the SUB instruction involving r can be implemented using a much simpler SUB module as shown in Fig. 5. It requires only one auxiliary neurons since the spike coming out of σ_r will not go to any auxiliary neurons associated with the other SUB instructions involving r. We can see from Fig. 1 that L_r contains only one SUB instruction each for the registers 1, 2, 3 and 7. So, 16 auxiliary neurons can be saved for the four SUB instructions involving these four registers.

Having the result of the computation in register 8, we can output the result by means of the module $OUTPUT$ in Fig. 6.

When neuron l_h receives a spike, it fires and sends a spike to neurons σ_8, σ_{b_8}, $\sigma_{l_{h,1}}$, $\sigma_{l_{h,2}}$ and $\sigma_{l_{h,3}}$. Let t be the moment when neuron l_h fires. Suppose the number stored in the register 8 of M'_u is n.

If σ_8 has no anti-spikes (when the value in register 8 is zero), at step $t+1$, five neurons $\sigma_{l_{h,1}}$, $\sigma_{l_{h,2}}$, $\sigma_{l_{h,3}}$, σ_{b_8} and σ_8 fire and all of them spike immediately except σ_{b_8} which sends an anti-spike. Neurons σ_{out} and σ_{b_8} are left with two spikes in each. So the computation halts without any spike emitted to the environment, representing the number 0 as the result.

If σ_8 has $n > 0$ anti-spikes (when the value of register 8 is $n > 0$), we can observe from the Fig. 6 that at the step $t + 1$, only three neurons $\sigma_{l_{h,1}}$, $\sigma_{l_{h,2}}$, $\sigma_{l_{h,3}}$ (neurons σ_8 and σ_{b_8} will not fire as the incoming spike is annihilated with one of their anti-spikes) fire. Neuron $\sigma_{l_{h,1}}$ sends a spike to σ_{out} and in the step

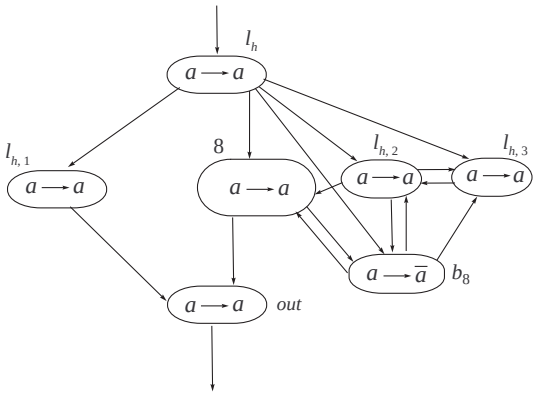

Fig. 6. Module $OUTPUT$

$t+2$ neuron σ_{out} fires for the first time sending 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 neurons σ_8 and σ_{b_8}. The neuron σ_8 does not fire until it has any anti-spikes. This means that the process of removing anti-spikes from the neuron σ_8 and σ_{b_8} continues, iteratively having neuron $\sigma_{l_{h,2}}$ sending spikes until σ_8 and σ_{b_8} have no anti-spikes. Thus neurons σ_8 and σ_{b_8} fire at the step $t+n+1$ for the first time when their contents become empty. Neuron σ_8 sends a spike to σ_{out}, σ_{b_8}. Neurons $\sigma_{l_{h,2}}$ and $\sigma_{l_{h,3}}$ become empty as spikes in them get annihilated with the anti-spike from σ_{b_8}. At step $t+n+2$, neuron σ_{b_8} is left with two spikes and the computation halts with a second spike coming out of neuron σ_{out}. The interval between the two spikes emitted by σ_{out} is $(t+n+2)-(t+2)=n$, which is exactly the number stored in the register 8 of M'_u.

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 18 neurons for the 9 registers, 25 neurons for the 25 labels, 10 neurons for the 10 ADD instructions, 54 neurons for 14 SUB instructions, 9 neurons in the $INPUT$ module and 4 neurons in the $OUTPUT$ module, which comes to a total of 120 neurons. This number can be slightly decreased, by some "code optimization", exploiting some particularities of the register machine M'_u.

We can simulate the consecutive $ADD-SUB$ instructions $l_5 : (ADD(5), l_6)$, $l_6 : (SUB(7), l_7, l_8)$ by merging the ADD module of l_5 with the SUB module of l_6. The merging can be performed by removing the neuron σ_{l_6} from the ADD and the SUB modules and adding outgoing synapses from the neuron σ_{l_5} to neurons σ_7, σ_{b_7} and $\sigma_{l_{6,1}}$. 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.

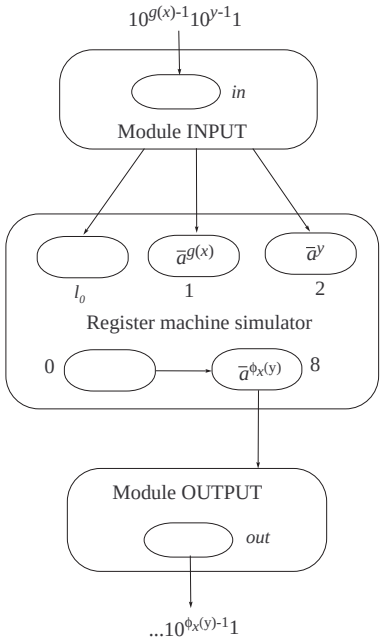

Fig. 7. The general design of the universal SN PA system

A similar operation is possible for the following four sequences of $SUB-ADD$ instructions, where we can save the intermediate labels (l_5, l_9, l_{16}, l_{22}), as well as one auxiliary neuron for each pair:
$l_4 : (SUB(6), l_5, l_3)$, $l_5 : (ADD(5), l_6)$,
$l_8 : (SUB(6), l_9, l_0)$, $l_9 : (ADD(6), l_{10})$,
$l_{14} : (SUB(5), l_{16}, l_{17})$, $l_{16} : (ADD(4), l_{11})$,
$l_h : (SUB(0), l_{22}, l'_h)$, $l_{22} : (ADD(8), l_h)$.

The module for the first pair is shown in Fig. 8. For each pair, we save two neurons; so eight neurons are saved for the four sequences.

Here we can also save neurons in $SUB - ADD$ instructions with the label of the second instruction used only in the else part of the first instruction. Let us consider the instructions $l_{14} : (SUB(5), l_{16}, l_{17})$, $l_{17} : (ADD(2), l_{21})$ and $l_{21} : (ADD(3), l_{18})$, where l_{17} is used only in the else part of l_{14}. Instructions l_{17} and l_{21} are two consecutive ADD instructions without any other instruction addressing the label l_{21}. The module for this set of three instructions can be constructed by merging the modules corresponding to these instructions. Four neurons (associated with auxiliary neurons and labels l_{17} and l_{21}) are saved by deleting them and adding synapses from the neuron $\sigma_{l_{14,5}}$ to neurons σ_2, σ_3, σ_{b_2} and σ_{b_3} and from the neuron $\sigma_{l_{14,2}}$ to $\sigma_{l_{18}}$. Similarly we can save two neurons for the pair of the instructions $l_{15} : (SUB(3), l_{18}, l_{20})$, $l_{20} : (ADD(0), l_0)$ (here

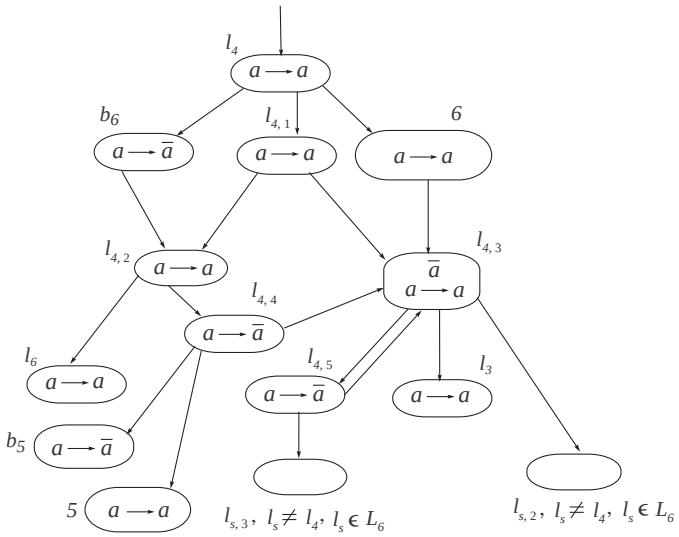

Fig. 8. A module simulating $SUB - ADD$ instructions

σ_{b_r} is connected with the neurons of ADD module). So six neurons are saved in these two cases.

If we consider all the above discussed optimizations, we save a total of 16 neurons and get the improvement from 120 to 104 neurons. We state this result in the form of a theorem in order to stress its importance:

Theorem 1. *There exists a universal computing spiking neural P system with anti-spikes having 104 neurons with each neuron having only one rule of the form $a \to a$ or $a \to \overline{a}$.*

4 Conclusion

A universal computing SN PA system was designed by T. Song et al. using 75 neurons with a total of 125 rules, 6 different types of neurons and 8 different types of rules. In this work, the problem of constructing universal SN PA systems with less number and types of rules is investigated. The systems constructed in this work has 104 simple neurons with rules of the form $a \to a$ or $a \to \overline{a}$. This improvement in the number and types of rules of the small universal SN PA system given by Theorems 1 is due to the use of two neurons with different rules to represent each register. It is possible to use less number of neurons to construct universal SN PA systems provided that neurons have more types of spiking rules.

Acknowledgements The work was supported by the European Regional De-

velopment Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/ 1.1.00/02.0070).

References

1. A. Păun and Gh. Păun: Small universal spiking neural P systems, *BioSystems*, **90**, 48–60 (2007).
2. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: Handbook of Membrane Computing. Oxford University Press, (2010).
3. G. Rozenberg, A. Salomaa, eds.: Handbook of Formal Languages. 3 volumes, Springer, Berlin, (1998).
4. I. Korec: Small universal Turing machines, *Theoretical Computer Science*, **168**, 267–301 (1996).
5. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes, *International Journal of Computers, Communications and Control*, **4**(3), 273–282 (2009).
6. M. Ionescu, Gh. Păun and T. Yokomori: Spiking neural P systems, *Fundamenta Informaticae*, **71**, 279-308 (2006).
7. M. Minsky: Computation – Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, (1967).
8. The P System Web Page: *http://ppage.psystems.eu*.
9. T. Song, L. Pan, J. Wang, I. Venkat, K.G. Subramanian, and R. Abdullah: Normal forms of spiking neural P systems with anti-spikes, *IEEE Transactions on Nanobioscience*, **11**(4), 352–359 (2012).
10. T. Song, Y. Jiang, X. Shi, and X. Zeng: Small universal spiking neural P systems with anti-spikes, *Journal of Computational and Theoretical Nanoscience*, **10** (4), 999–1006 (2013).
11. X. Zeng, X. Zhang and L. Pan: Homogeneous spiking neural P systems, *Fundamenta Informaticae*, **97** (1–2), 1–20 (2009).