New Mathematics and Natural Computation
Vol. 8, No. 3 (2012) 283–295
© World Scientific Publishing Company
DOI: 10.1142/S1793005712500032

COMPUTABILITY OF SPIKING NEURAL P SYSTEMS WITH ANTI-SPIKES

VENKATA PADMAVATI METTA*

Department of Computer Applications
Bhilai Institute of Technology
Durg, Chhattisgarh, India
vmetta@gmail.com

KAMALA KRITHIVASAN

Department of Computer Science and Engg.
Indian Institute of Technology
Chennai, Tamilnadu, India
kamala@iitm.ac.in
www.cse.iitm.ac.in∼kamala/

DEEPAK GARG

Department of Computer Science and Engg.
Thapar University, Patiala, Punjab, India
deep108@yahoo.com

Spiking neural P systems with anti-spikes (for short, SN PA systems) can encode the binary digits in a natural way using two types of objects called anti-spikes and spikes. In this paper, we use SN PA systems to perform the arithmetic operation like 2's complement, addition and subtraction of binary numbers. They are also used to simulate NAND and NOR gates.

Keywords: Spiking neural P system with anti-spikes; simulation.

1. Introduction

It is obvious that the chemical, electrical and informational processes taking place in the brain are the major source of inspiration for informatics. Risking a forecast, we believe that if something great is to appear in informatics in the near future, then it will be inspired by the brain.

Spiking neural P system[5] (shortly called SN P system) is a parallel and distributed computing model inspired by the neurobiological behavior of neurons sending electrical pulses of identical voltages called spikes to neighboring neurons. It is a versatile formal model of computation that can be used to design efficient parallel algorithms

*Padmavati Metta, Bhilai Institute of Technology, Durg, India.

for solving known computer science problems. Spiking neural P systems are not the answer to this learning-from-brain challenge, but only to call (once again) the attention to this challenge. Becoming familiar with brain functioning, in whatever reductionistic framework (as spiking neural P systems investigation is), can however be useful.

Standard SN P system is pictorially represented as a directed graph where nodes correspond to neurons having spiking and forgetting rules. The rules involve the spikes present in the neuron in the form of occurrences of a symbol a. The arcs indicate the synapses among the neurons. Similar to the neurons in the brain, the neurons in an SN P system also fire in parallel, with each neuron using only one rule in each step. The initial configuration of the system is represented by the number of spikes present in each neuron. One of the neuron is taken as output neuron, which sends spikes to environment. Number of outputs can be associated with an SN P system. Hence, in a standard SN P system there are only one type of objects called spikes which are moved, created and destroyed but never modified into another form.

Standard spiking neural P systems are used to simulate arithmetic and logic operations where the presence of spike is encoded as 1 and absence of spike as 0 and the negative integers were not considered.[3] The ability of SN P systems to efficiently simulate Boolean circuits are studied,[4] since apart from being a well known computational model, there exist many "fast" algorithms solving various problems. In addition, this simulation, enriched with some "memory modules" (given in the form of some SN P sub-systems), may constitute an alternative proof of the computational completeness of the model. The Boolean value 1 is encoded in the SN P system by two spikes, hence a^2, while 0 is encoded as one spike. As the system has only one input neuron, the number of spikes equal to the sum of the inputs, is introduced into the neuron. For example to compute the logical AND or OR operation between 1 and 0 (or 0 and 1) three spikes (two spikes for 1 and one spike for 0) are introduced into the input neuron and four spikes are introduced for case 11.

SN P system with anti spikes,[2] is a variant of an SN P system consisting of two types of objects, spikes (denoted as a) and anti-spikes (denoted as $\bar{a}$). The inhibitory impulses/spikes are represented using anti-spikes. The anti-spikes behave in a similar way as spikes by participating in spiking and forgetting rules. They are produced from usual spikes by means of usual spiking rules; in turn, rules consuming anti-spikes can produce spikes or anti-spikes (here we avoid the rule anti-spike producing anti-spike). The SN P system with anti-spikes consists of an implicit annihilation rule of the form $a\bar{a} \to \lambda$; if an anti-spike and a spike meet in a given neuron, they annihilate each other. This rule has the highest priority and does not consume any time. SN P system with anti-spikes allows the modification of spikes and anti-spikes and is proved as computationally complete.

In this paper, we use SN P systems with anti-spikes to simulate logic gates. The advantage of using this variant of SN P system is that we can encode 1 by a spike and 0 by an anti-spike. A Boolean gate can be simulated in a very natural way using two inputs and one output. Input data can be introduced into the system in a similar way to any Boolean gate. Using SN P system with anti-spikes, we can perform the operations on negative numbers also. The input to the systems is a binary sequence of spikes and anti-spikes which encodes the digits 1 and 0 respectively, of a binary number. They can represent the negative numbers in 2's complement form, thereby simulating the arithmetic operations on negative numbers. In this paper, we have simulated three arithmetic operations — 2's complement, addition and subtraction.

1.1. *Notation*

We recall here a few definitions and notations related to the formal languages and automata theory.

Σ is a finite set of symbols called alphabet. A string w over Σ is a sequence of symbols drawn from Σ. λ denotes the empty string. Σ^* is the set of all strings over Σ. $\Sigma^* - \{\lambda\}$ is denoted by Σ^+. The length of a string w is denoted by $|w|$. A language L over Σ is a set of strings over Σ.

A language $L \subseteq \Sigma^*$ is said to be regular if there is a regular expression E over Σ such that $L(E) = L$. The regular expressions are defined using the following rules. (i) ϕ, λ and each $a \in \Sigma$ are regular expressions. (ii) if E_1, E_2 are regular expressions over Σ, then $E_1 + E_2$, E_1E_2 and E_1^* are regular expressions over Σ, and (iii) nothing else is a regular expression over Σ. With each regular expression E, we associate a language $L(E)$.

When $\Sigma = \{a\}$ is a singleton, then the regular expression a^* denotes the set of all strings formed using a. i.e. the set $\{\epsilon, a, a^2, a^3, \ldots\}$. The positive closure $a^+ = a^* - \{\lambda\}$.

2. Spiking Neural P System with Anti-Spikes

Here we recall the definition of SN P system with anti-spikes without delay.

Definition 2.1. (*SN P system with anti-spikes*) Mathematically, we represent a spiking neural P system with anti-spikes of degree $m \geq 1$, in the form

$$\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m, syn, i_0), \text{ where}$$

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

$$\sigma_i = (n_i, R_i), \quad 1 \leq i \leq m,$$

where

(a) $n_i \geq 0$ is the *initial number of spikes or anti-spikes* contained by the cell. Neuron σ_i has n_i spikes if $n_i > 0$ or n_i anti-spikes if $n_i < 0$.
(b) R_i is a finite set of *rules* of the following two forms:

(i) $E/b^r \to b'$ where E is a regular expression over a or $\bar{a}$, while $b, b' \in \{a, \bar{a}\}$, and $r \geq 1$.

(ii) $b^r \to \lambda$, for some $r \geq 1$, with the restriction that $b^r \notin \mathrm{L}(E)$ for any rule $E/b^r \to b'$ of type (1) from R_i;

(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) $i_0 \in \{1, 2, 3, \ldots, m\}$ indicates the *output neuron.*

The rules of type $E/b^r \to b'$ are spiking rules, and they are possible only if the neuron contains n spikes such that $b^n \in \mathrm{L}(E)$ and $n \geq r.\, l(v)$ and $r(v)$ gives the number of spikes/anti-spikes present in the left and right-hand sides of rule v respectively. Further $l(E/b^r \to b') = r$ if $b = a$ and $l(E/b^r \to b') = -r$ if $b = \bar{a}$. The value of $r(v)$ is either 1(if $b' = a$) or -1(if $b' = \bar{a}$). Here we avoid the use of rules of the form $\bar{a}^c \to \bar{a}$, but not the other three types, corresponding to the pairs (a, a), $(a, \bar{a})$, $(\bar{a}, a)$. If E is omitted then the rule is applied only if the neuron contains exactly r spikes/anti-spikes. When neuron σ_i sends spike/anti-spike, it is replicated in such a way that one spike/anti-spike is sent to all neurons σ_j such that $(i, j) \in syn$. There is an additional fact that a and $\bar{a}$ cannot stay together, so annihilate each other. If a neuron has either objects a or objects $\bar{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^r and $\bar{a}^s$ inside, then immediately a rule of the $a\bar{a} \to \lambda$, which is implicit in each neuron, is applied in a maximal manner, so that either a^{r-s} or $\bar{a}^{s-r}$ remain for the next step, provided that $r \geq s$ or $s \geq r$, respectively. This mutual annihilation of spikes and anti-spikes takes no time and the rule has the highest priority.

The rules of type $b^r \to \lambda$ are forgetting rules; r spikes are simply removed ("forgotten") when applying. Like in the case of spiking rules, the left-hand side of a forgetting rule must "cover" the contents of the neuron, that is, $a^s \to \lambda$ is applied only if the neuron contains exactly s spikes.

The simple SN P system works in a similar way but with only one type of object called *spike*(a) and so there exist no annihilation rules.

Definition 2.2. (*Configuration*) The configuration of the system is described by the number of spikes/anti-spikes present in each neuron. Thus $\langle n_1, n_2, \ldots, n_m \rangle$ is a configuration where neuron σ_i, $i = 1, 2, 3, \ldots, m$ contains n_i spikes if $n_i > 0$ or n_i anti-spikes if $n_i < 0$.

A global clock is assumed in SN P system 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 except the annihilation rule which fires maximally with highest priority. For example, if a neuron σ_i has two firing rules, $E_1/a^r \to a$ and $E_2/a^k \to a$ with $L(E_1) \cap L(E_2) \neq \emptyset$, then it is possible that two can be applied in a neuron, 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. The rules are used in the non-deterministic manner, in a maximally parallel way at the level of the system; in each step, all neurons which can use a rule of any type, spiking or forgetting, have to evolve, using a rule.

Definition 2.3. (*Vector rule*) We define a vector rule v as a mapping with domain Π such that each $v(i)$ is at most one instance of spiking or forgetting rule from R_i i.e. $|v(i)| = 0$ or 1 where $1 \leq i \leq m$.

If a vector rule v is enabled at a configuration $\mathcal{C} = \langle n_1, n_2, \ldots, n_m \rangle$ then $\mathcal{C}$ can evolve to $\mathcal{C}' = \langle n'_1, n'_2, \ldots, n'_m \rangle$ (after applying annihilation rules in each neuron in maximal way), where $n'_i = n_i - l(v(i)) + \sum_{(j,i) \in syn} r(v(j))$.

Definition 2.4. (*Transition*) Using the vector rule, we pass from one configuration of the system to another configuration, such a step is called a transition. For two configurations C and C' of Π we denote by $C \Rightarrow C'$, if there is a direct transition from C to C' in Π.

A computation of Π is a finite or infinite sequences of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. Note that the transition of C is non-deterministic in the sense that there may be different vector rules applicable to C, as described above.

A computation halts if it reaches a configuration where no rule can be used. There are various ways of using such a device.[1] In the generative mode, one of the neurons is considered to be the output neuron, and its spikes are sent to the environment. With any computation halting or not we associate a spike train, a sequence of digits of 0 and 1, with 1 and 0 appearing in positions which indicate the steps when the output neuron sends spikes and anti-spikes respectively, out of the system. With any spike train we can associate various numbers which are considered as computed by the system. Because of the non-determinism in using the rules, a given system computes in this way a set of numbers. When both an input and an output neuron are considered, the system can be used as a transducer, both for strings and infinite sequences, as well as for computing numerical functions. Spikes can be introduced in the former one, at various steps, while the spikes of the output neuron are sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the moments of anti-spikes are marked with 0. The binary sequence obtained in this way is called the spike train of the system; it might be infinite if the computation does not stop. A binary sequence is similarly associated with the spikes entering the system. In the transducing mode, a large class of (Boolean) functions can be computed.

Example 2.1. Consider the graphical representation of an SN P system with anti-spikes in Fig. 1. The neurons are represented by nodes of a directed graph whose

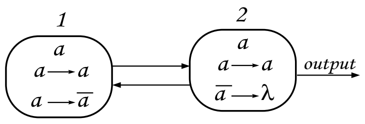

Fig. 1. SN P system with anti-spikes generating 1^+.

arrows represent the synapses; an arrow also exits from the output neuron, pointing to the environment; in each neuron we specify the rules and the spikes present in the initial configuration. It is formally denoted as

$$\begin{aligned}\Pi_1 &= (O, \sigma_1, \sigma_2, syn, 2), \text{ with}\\ \sigma_1 &= (1, \{a \to a, a \to \bar{a}\}),\\ \sigma_2 &= (1, \{a \to a, \bar{a} \to \lambda\}),\\ syn &= \{(1,2),(2,1)\}.\end{aligned}$$

We have two neurons, with labels 1, 2; neuron 2 is the output neuron. Initially neuron 1 has one spike with non-determinism between its two rules and neuron 2 has one spike and they fire in the first step. Neuron 2 uses its first rule and sends a spike (1) to environment and neuron 1. Neuron 1 can choose any of its two rules and as long as it uses first rule, one spike will be sent to neuron 2, which uses first rule in the next step by sending a spike to environment and neuron 1. At any instance of time, starting from step 1, neuron 1 can choose its second rule, which modifies spike into anti-spike and sent to neuron 2. The anti-spike is ignored by the neuron 2 in the next step (using rule 2) and the system halts. As the neuron 2 emits a spike in the first step, even if the neuron 1 uses the second rule in the first step, at least one spike (1) is emitted by the system. Because of the non-determinism in using the rules of neuron 1, the system computes a set of binary strings (spike train) represented using regular expression 1^+.

3. Simulating Universal Logic Gates

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates. The NAND gate represents the complement of the AND operation and the NOR gate represents the complement of the OR operation. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families. In this section, we simulate the NAND and NOR gates using SN P systems with anti-spikes working in transducing mode. The Boolean values 0 and 1 are encoded in the SN P system by anti-spike and spike respectively. The output of the system is 0(hence false) if the output neuron sends an anti-spike and output is 1(true) if a spike is sent to the environment. We want to emphasize that no rule of the form $\bar{a}^c \to \bar{a}$ is used.

Lemma 3.1. *Boolean NAND and NOR gates can be simulated by SN PA systems with three neurons in two steps.*

Proof. We construct SN PA system with seven neurons as in Fig. 2. The SN PA system has two input neurons to take the input values and one output neuron to produce output. A spike/anti-spike is introduced in each input neuron corresponding to input 1/0.

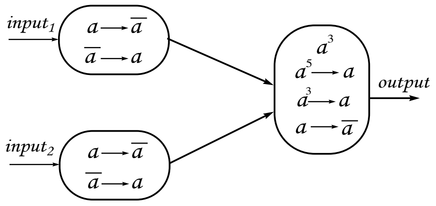

Fig. 2. SN P system with anti-spikes simulating 2-input NAND gate.

If we introduce an anti-spike (0) into each of the input neurons, the anti-spike becomes a spike and sent to the output neuron in the next stage. So the output neuron gets two spikes from the input neurons and it already has three spikes, accumulating a total of five spikes and fires using a rule $a^5 \rightarrow a$ sending a spike (1) to the environment. But if we introduce a spike (1) into each of the input neurons, the output neuron gets two anti-spikes and gets annihilated with two spikes already present in it, remains with a spike and fires using a rule $a \rightarrow \bar{a}$ producing an anti-spike (0). In the third case, if a spike is introduced into one of the input neurons and an anti-spike into another, then they get annihilated after reaching the output neuron. So the output neuron has its three spikes and fires using the rule $a^3 \rightarrow a$ sending a spike to the environment. We can observe that it is simulating the NAND gate correctly.

If we replace the rule $a^3 \rightarrow a$ with $a^3 \rightarrow \bar{a}$ in the output neuron of the above system, we obtain the SN PA system for the NOR gate.

Similar to the 2-input NAND gate, we can construct n-input NAND gate. The output of the gate is false (0) only if all the inputs are true (1) and is true if any of the inputs is false. The SN PA system for n-input NAND gate is shown in Fig. 3. The maximum number of anti-spikes received by the output neuron is n (if all inputs are spikes corresponding to true) and they get annihilated with n spikes in the output neuron and is left with a spike and fires using the rule $a \rightarrow \bar{a}$ producing an anti-spike.

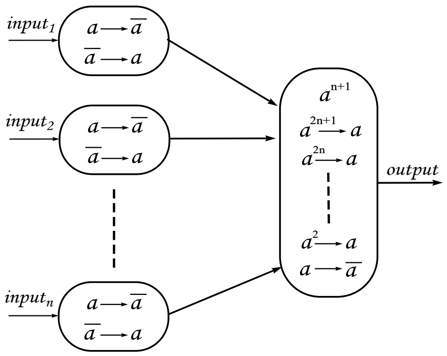

Fig. 3. SN P system with anti-spikes simulating n-input NAND gate.

In all other cases, it produces a spikes. Thus simulating the n-input NAND gate correctly. $\square$

4. Simulating Circuits

Here, we present the way to simulate any Boolean circuit using NAND or NOR gates constructed in the previous section. We know that any Boolean function can be represented in sum-of-product (SOP) and product-of-sum forms (POS). SOP forms can be implemented using only NAND gates, while POS forms can be implemented using only NOR gates. In either case, implementation requires two levels. The first level is for each term and second level for product or sum of the terms.

Consider the Boolean function $\neg(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$. It is written in SOP from as $\neg x_1 \vee \neg x_2 \vee (x_3 \wedge x_4)$.

We use the SN P systems with anti-spikes for 2-input and 3-input NAND gates. Let Π_{NAND} is an SN P systems for NAND gate. The Boolean circuit corresponding to the above formula as well as the spiking system assigned to it are depicted in Fig. 4.

Note that in Fig. 4, $\Pi^{(1)}_{NAND}, \Pi^{(2)}_{NAND}, \Pi^{(3)}_{NAND}$ are SN P systems for 2-input NAND gates and $\Pi^{(4)}_{NAND}$ is the SN P system for 3-input NAND gate. Having the overall image of the functioning of the system, let us give some more details on the simulation of the above formula. For that we construct the SN P system with anti-spikes $\Pi_C = (\Pi^{(1)}_{NAND}, \Pi^{(2)}_{NAND}, \Pi^{(3)}_{NAND}, \Pi^{(4)}_{NAND})$ formed by the sub-SN P systems for each gate and we obtain the unique result as follows:

(1) For every gate of the circuit with inputs from the input gates we have a SN P system to simulate it. The input is given to the input neurons of each gate;
(2) For each gate which has at least one input coming as an output of a previous gate, we construct a SN P system to simulate it by adding a synapse from the output neuron of the gate from which the signal (spike) comes to the input neuron of the system that simulates the new gate.

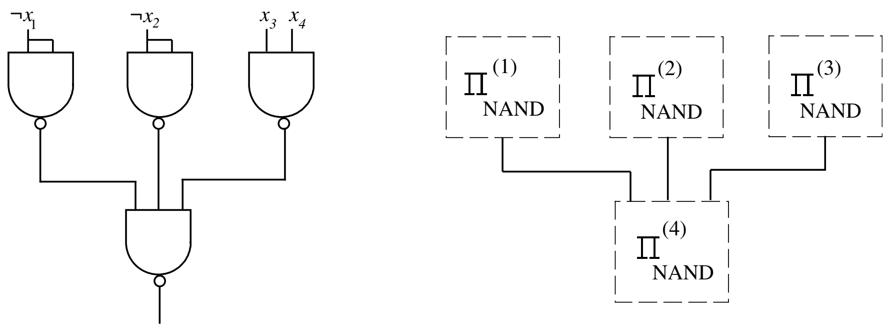

Fig. 4. Boolean circuit and corresponding SN P system with anti-spikes for $\neg(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$.

For the above formula and the circuit depicted in Fig. 4, we will have:

$\Pi^{(1)}_{NAND}$ performs the first NAND operation $\neg(\neg x_1 \wedge \neg x_1) = x_1$ with each input as $\neg x_1$. (For $\neg x_1$ as input, an anti-spike is introduced in each input neuron of $\Pi^{(1)}_{NAND}$).
$\Pi^{(2)}_{NAND}$ performs the second NAND operation $\neg(\neg x_2 \wedge \neg x_2) = x_2$ with each input as $\neg x_2$.

$\Pi^{(3)}_{NAND}$ performs the third NAND operation $\neg(x_3 \wedge x_4)$ with inputs as x_3 and x_4. These three SN P systems $\Pi^{(1)}_{NAND}$, $\Pi^{(2)}_{NAND}$ and $\Pi^{(3)}_{NAND}$ act in parallel producing the output at the same time. The outputs enter the 3-input NAND gate $\Pi^{(4)}_{NAND}$ at the same time which eliminates the use of synchronising module.[4]

$\Pi^{(4)}_{NAND}$ computes NAND operation on x_1, x_2 and $\neg(x_3 \wedge x_4)$ outputting $\neg x_1 \vee \neg x_2 \vee (x_3 \wedge x_4)$ to the environment.

Generalizing the previous observations the following result holds:

Theorem 4.1. *Every Boolean circuit can be simulated by an SN PA system and is constructed from SN P systems with anti-spikes of type NAND or NOR, by reproducing the structure associated with the circuit.*

5. Arithmetic Operations using SN P System with Anti-Spikes

In this section, we consider SN P system with anti-spikes as simple arithmetic device that can perform the arithmetic operations like 2's complement, addition and subtraction with input and output in binary form. The binary sequence of 0 and 1 are encoded as anti-spike and spike respectively and in each time step input is provided bit-by-bit starting from the least significant bit. The negative numbers are represented in two's complement form. The advantage of using SN P systems with anti-spikes is that they can encode the 0 and 1 as anti-spike and spike in a very natural way and thus providing a way to represent negative numbers also.

5.1. *2's Complement*

The 2's complement is used to represent a negative of a binary number. It also gives us a straightforward way to add and subtract positive and negative binary numbers. A simple way to find the 2's complement of a number is to start from the least significant bit keeping every 0 as it is until you reach the first 1 and then complement all the rest of the bits after the first 1.

Theorem 5.1. 2*'s complement of a binary number can be calculated using an SN P systems with anti-spikes using three neurons.*

Proof. The SN P system that performs the 2's complement is shown in Fig. 5. Neuron 1 is the input neuron. Neuron 3 is the output neuron, which sends output to environment. The input neuron has two rules to complement the input by changing a spike into anti-spike and anti-spike into spike and send it to its neighboring neuron 2. The neuron 2 initially has 3 spikes and as long as it receives a spike (actual input to

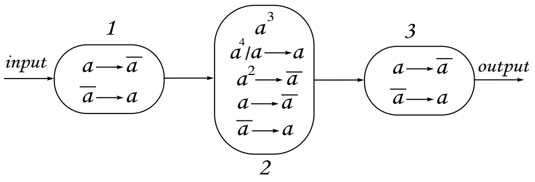

Fig. 5. SN P system computing 2's complement.

the input neuron is 0), it uses the first rule $a^4/a \to a$ by send a spike to the output neuron where it is complemented into anti-spike, which is same as the input. But if the second neuron receives anti-spike (that means we got the first 1), it will be left with two spikes because of the annihilation rule that is implicitly present in each neuron and uses the rule $a^2 \to \bar{a}$ and sends an anti-spike to the output neuron where it is complemented as spike and sent to the environment (that is first 1 is unchanged). After firing the rule, the neuron 2 has no spikes/anti-spikes and then simply complements the input it receives by using the third and fourth rule and sends it to the output neuron where it is again complemented and sent to environment. That means after the first one, the output will be the complement of input. We can easily observe that the system correctly calculates the 2's complement and emits its first output bit at $t = 4$ as there is one intermediate neuron. □

As an example, let us consider a binary number 01100 (12 in decimal). The way the SN P system computes the 2's complement is represented in Table. 1. It reports the number of spikes/anti-spikes present in each neuron and output produced by the output neuron to the environment in the output column.

5.2. *Addition and subtraction*

The SN P system performing the addition is shown in Fig. 6. The negative numbers are represented in 2's complement form using the system SN P system given in the previous section and then fed as input.

Table 1. Number of spikes/anti-spikes present in each neuron of an SN P system during the computation of 2's complement of 01100.

Time	Input	Neuron 2	Neuron 3	Output
$t = 0$	—	a^3	—	—
$t = 1$	$\bar{a}(0)$	a^3	—	—
$t = 2$	$\bar{a}(0)$	a^4	—	—
$t = 3$	$a(1)$	a^4	a	—
$t = 4$	$a(1)$	a^2	a	$\bar{a}(0)$
$t = 5$	$\bar{a}(0)$	$\bar{a}$	$\bar{a}$	$\bar{a}(0)$
$t = 6$	—	a	a	$a(1)$
$t = 7$	—	—	$\bar{a}$	$\bar{a}(0)$
$t = 8$	—	—	—	$a(1)$

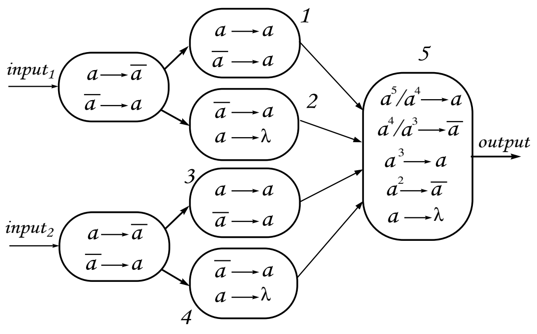

Fig. 6. An SN P system with anti-spikes simulating addition operation.

Theorem 5.2. *Addition of two binary numbers can be performed using an SN P systems with anti-spikes.*

Proof. The system has two input neurons, the first number is provided through input neuron 1 and the second one is through input neuron 2. Input neuron 1 is connected to neurons 1 and 2 and input neuron 2 is connected to neurons 3 and 4. The presence of a spike in the output neuron indicates a carry of the previous addition. Each input neuron has two rules to complement the input and send the output to its neighboring two neurons. Here we are having 3 cases:

(1) If both the inputs are 1(spike), then in each input neuron uses the second rules and sends an anti-spike two of its neighboring neurons where the anti-spikes are converted spikes. So the output neuron 5 receives four spikes, one from each of the four neurons of the previous stage. If the output neuron is already having a spike(carry), then the number of spikes become 5 and fires using a rule $a^5/a^4 \to a$ otherwise it has four spikes and fires using the rule $a^4/a^3 \to \bar{a}$ leaving one spike in the output neuron in either case. The presence of a spike in the output neuron indicates a carry. This encodes the two operations $1 + 1 = 0$ with carry 1 and $(1) + 1 + 1 = 1$ with carry 1.

(2) If one of the input bit is zero, then the input neuron receiving an anti-spike sends a spike to each of it's neighboring neurons. For example if the input 1 is 0 and input 2 is 1 then input neuron 1 sends a spike two each of neighboring neurons 1 and 2. In the neuron 1, the spike remain the same and where as in neuron 2 it is forgotten, so the number of spikes sent to the output neuron is 1, where as the neighboring neurons of input neuron 2 sends two spikes to the output neuron. So three spikes are received if one of the input is zero. The output neuron has either three or four (in case carry) spikes and fires using $a^3 \to a$ or $a^4/a^3 \to \bar{a}$ respectively. These rules encode the two operations $0 + 1 = 1$ and $(1) + 0 + 1 = 0$ with carry 1 respectively.

(3) If both the input neurons receive anti-spikes(0), then the output neuron receives two spikes and it will have either two or three (again in case of carry of the

Table 2. Number of spikes/anti-spikes present in each neuron of addition SN P system during the addition of 0111 and 1011.

Time	Input 1	Input 2	σ_1	σ_2	σ_3	σ_4	σ_5	Output
$t = 1$	$a(1)$	$a(1)$	—	—	—	—	—	—
$t = 2$	$a(1)$	$a(1)$	$\bar{a}$	$\bar{a}$	$\bar{a}$	$\bar{a}$	—	—
$t = 3$	$a(1)$	$\bar{a}(0)$	$\bar{a}$	$\bar{a}$	$\bar{a}$	$\bar{a}$	a^4	—
$t = 4$	$\bar{a}(0)$	$a(1)$	$\bar{a}$	$\bar{a}$	a	a	a^5	$\bar{a}(0)$
$t = 5$	—	—	a	a	$\bar{a}$	$\bar{a}$	a^4	$a(1)$
$t = 6$	—	—	—	—	—	—	a^4	$\bar{a}(0)$
$t = 7$	—	—	—	—	—	—	a	$\bar{a}(0)$
$t = 8$	—	—	—	—	—	—	—	—

previous operation) spikes and fires using $a^2 \rightarrow \bar{a}$ or $a^3 \rightarrow a$. These two rules do not leave any carry encoding the operations $0 + 0 = 0$ and $(1) + 0 + 0 = 1$ respectively.

The last rule in the output neuron $a \rightarrow \lambda$ allows the last overflow bit to be ignored. The procedure confirms the correctness of the system for performing the addition of two numbers. $\square$

As an example, let us consider the addition of 7 and -5. Number 7 is represented in binary form as 0111 and -5 is represented in 2's complement form as 1011. The two binary sequences will form the input for the SN P system. The number of spikes present in each neuron in every step and the output produced by the system is depicted in Table. 2.

Two's complement subtraction is the binary addition of the minuend to the 2's complement of the subtrahend (adding a negative number is the same as subtracting a positive one). That means $a - b$ becomes $a + (-b)$. The SN P system for addition can be used to perform subtraction. The multiplication is viewed as repeated addition and division as repeated subtraction. This implies that SN P systems with anti-spikes can very well perform the binary operations in a natural way.

6. Conclusion

In this paper, we designed SN P systems with anti-spikes to perform arithmetic operations like 2's complement, addition and subtraction. The advantage of using this variant of SN P system is that spikes and anti-spikes can encode the binary digits in a more natural way and we can perform the operations on negative numbers also. The input to the systems is a binary sequence of spikes and anti-spikes which encodes the digits 1 and 0 respectively, of a binary number. The negative numbers are in 2's complement form. The outputs of the computations are also expelled to the environment in the same form. We also designed SN PA systems simulating the operations of NAND and NOR gates. This motivates the implementation of CPU using SN P systems with anti-spikes.

References

1. Gh. Păun, Spiking neural P systems used as acceptors and transducers, *CIAA*, *LNCS* **4783**(1–4) (2007) 1–4.
2. L. Pan and Gh. Păun, Spiking neural P systems with anti-spikes, *Int. J. of Computers, Communications and Control* **4**(3) (2009) 273–282.
3. M. A. Gutiérréz-Naranjo and A. Leporati, First steps towards a CPU made of spiking neural P systems, *Int. J. of Computers, Communications and Control* **4** (2009) 244–252.
4. M. Ionescu and D. Sburlan, Some applications of spiking neural P systems, *J. of Computing and Informatics* **27** (2008) 515–528.
5. M. Ionescu, Gh. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* **71**(2–3) (2006) 279–308.