ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **14**, Number 1, 2011, 20–30

Spiking Neural P Systems with Anti-Spikes as Transducers

Venkata Padmavati METTA[1],
Kamala KRITHIVASAN[2], Deepak GARG[3]

[1]Bhilai Institute of Technology, Durg, India
[2]Indian Institute of Technology, Chennai, India
[3]Thapar University, Patiala, India

Abstract. In this paper, we consider spiking neural P systems with anti-spikes. Because of the use of two types of objects, the system can encode the binary digits in a natural way and hence represent the formal models more efficiently and naturally than the standard SN P systems. This work deals with the computing power of spiking neural P system with anti-spikes. It is demonstrated that, as transducers, spiking neural P systems with anti-spikes can simulate any Boolean circuit and also computing devices such as finite automata and finite transducers. We also investigate how the use of anti-spikes in spiking neural P systems affect the capability to solve the satisfiability problem.

1. Introduction

Spiking neural P systems [3] (shortly called SN P systems) are parallel and distributed computing models inspired by the neurophysiological behaviour of neurons sending electrical pulses of identical voltages called spikes to the neighbouring neurons through synapses. A standard SN P system is represented as a directed graph where nodes correspond to the neurons having spiking and forgetting rules that involve the spikes present in the neuron in the form of occurrences of a symbol a. The arcs indicate the synapses among the neurons. The spiking rules are of the form $E \,/\, a^r \rightarrow a$ and are used only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$, where $L(E)$ is the language represented by the regular expression E. In this case r spikes are consumed and one spike is sent out. When neuron σ_i sends a spike, it is replicated in such a way that one spike is immediately sent to all neurons σ_j such

that $(i, j) \in syn$, where syn is the set of arcs between the neurons. The transmission of spikes takes no time, the spike will be available in neuron σ_j in the next step. The forgetting rules are of the form $a^s \to \lambda$ and are applied only if the neuron contains exactly s spikes. The rule simply removes s spikes. For all forgetting rules, s must not be the member of $L(E)$ for any firing rule within the same neuron. A neuron is *bounded* if for every firing rule $E \,/\, a^r \to a$, E denotes a finite regular expression. An SN P system is called *bounded* if all the neurons in the system are *bounded*.

SN P systems with anti-spikes (for short, SN PA systems) introduced in [7], are a variant of SN P systems containing two types of objects, spikes (denoted by a) and anti-spikes (denoted by $\overline{a}$), corresponding somewhat to inhibitory impulses from neurobiology. 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 (here we avoid the rules producing anti-spikes from anti-spikes). An SN P system with anti-spikes also contains an implicit annihilation rule of the form $a\overline{a} \to \lambda$; 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. The initial configuration of the system is represented by the number of spikes/anti-spikes present in each neuron. A computation halts if it reaches a configuration where no rule can be used. There are various ways of using such a device: as an acceptor, generator, transducer [2].

In the generative mode, the system has no input neurons and one of the neurons is considered to be the output neuron, and its spikes are sent to the environment. The moments of time when spikes are emitted by the output neuron are marked with 1, the moments when anti-spikes are emitted are marked with 0 and no output steps are ignored. The binary sequence obtained in this way is called the spike train of the system. The binary strings describing the spike trains of the halting computations are taken as the language generated by the system. It was shown in [5] that some regular languages cannot be generated by spiking neural P systems with a bounded number of spikes in the neurons. As an SN PA system allows non-determinism between the rules $a^c \to a$ and $a^c \to \overline{a}$, $c \geq 1$, we can construct SN PA systems generating more languages than SN P systems [8].

When both input and output neurons 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 input neuron, at various steps, while the spikes of the output neuron are sent to the environment. A binary sequence is associated with the spikes/anti-spikes entering or exiting the system. In the transducer mode, a large class of (Boolean) functions can be computed.

Standard spiking neural P systems in transducer mode can simulate the Boolean circuits [4], with two spikes sent out of the system encoded as 1 and one spike as 0. In this paper we use SN P systems with anti-spikes to simulate logic gates, with the anti-spikes and spikes encoding the Boolean values 0 and 1 in the natural way. We design SN P systems with anti-spikes simulating the operations of AND, OR and NOT gates. The output of the system is 0 (hence false) if the output neuron sends out an anti-spike and 1 (true) if a spike is sent to the environment. Hence we present a way to simulate any Boolean circuit using these fundamental gates and synchronising

SN P system with anti-spikes to establish synchronization among the gates to output the correct result.

In [1], a uniform solution to the SAT (in CNF, with n variables and m clauses) is provided using standard SN P systems without delay having $3n^2 + 8m + 5$ neurons, providing the solution in a number of steps which is linear in the number of variables. Two bits were used to code each literal of a clause, hence the computation cannot end in less than $2n$ steps. Here we use only one bit to code each literal of clause C_j. 1 indicates the case when x_i appears in C_j, 0 indicates the case when $\neg x_i$ appears in C_j and λ (empty) indicates the absence of x_i in the clause C_j. So n bits are needed to code any clause. Using SN PA systems the number of steps can be reduced to half. These systems only requires $3m + 2$ neurons.

In this paper we first formally define SN PA systems. These are then used to construct Boolean circuits. Furthermore they are used to simulate computational devices such as finite state transducers. Finally we show how they can be used for solving the SAT problem.

2. Spiking Neural P System with Anti-Spikes

First we recall the definition of SN P systems with anti-spikes.

Definition 2.1. *(SN P systems with anti-spikes)* A spiking neural P system with anti-spikes, of degree $m \geq 1$, is a construct

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

1. $O = \{ a, \overline{a} \}$ is the binary alphabet. a is called *spike* and $\overline{a}$ is called *anti-spike*.

2. $\sigma_1, \sigma_2, \sigma_3 ,\cdots, \sigma_m$ are neurons, of the form

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

   (a) $\alpha_i = a^{n_i}$ or $\alpha_i = \bar{a}^{n_i}$, where n_i is the number of spikes or anti-spikes, respectively, contained 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 E is a regular expression over a or $\overline{a}$, while $b, b' \in \{a, \overline{a}\}$, and $r \geq 1$.

      ii. $b^s \rightarrow \lambda$,where λ is the empty word and $s \geq 1$, and for all $E/b^r \rightarrow b'$ from R_i, $b^s \notin L(E)$ where $L(E)$ is the language defined by E.

3. $syn \subseteq \{ 1, 2, 3, \cdots, m\} \times \{ 1, 2, 3, \cdots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among cells);

4. in, $out \in \{1, 2, 3, \cdots, m\}$ are the input and output neurons respectively.

The rules of type $E/b^r \to b'$ are spiking rules, and they are used only if the neuron contains n b's such that $b^n \in L(E)$ and $n \geq r$. When neuron σ_i sends b' (spike/anti-spike), it is replicated in such a way that one spike/anti-spike is sent to all neurons σ_j such that $(i, j) \in syn$.

The rules of type $b^s \to \lambda$ are forgetting rules; s spikes/anti-spikes are simply removed ("forgotten") when applying the rule. 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 or anti-spikes.

A spike/anti-spike emitted by neuron i will pass immediately to all neurons σ_j such that $(i, j) \in syn$. That means transmission of spikes/anti-spikes takes no time, the spikes/anti-spikes will be available in neuron σ_j in the next step. There is an additional fact 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^r 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^{r-s} or $\overline{a}^{s-r}$ 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 annihilation rule has priority over spiking and forgetting rules, so each neuron always contains either only spikes or anti-spikes. Like in [7], we avoid using rules $\overline{a}^c \to \overline{a}$, but not the other three types, corresponding to the pairs (a, a), $(a, \overline{a})$, $(\overline{a}, a)$. If we have a rule $E/b^r \to b'$ with $L(E) = \{b^r\}$, then we write it in the simplified form $b^r \to b'$.

The *configuration* of the system is described by $\mathcal{C} = \langle \beta_1, \beta_2, \cdots, \beta_m \rangle$ where β_i is the multiset written in the form $\beta_i = a^x \overline{a}^y$, where x is the number of spikes and y is the number of anti-spikes present in neuron σ_i and either $x = 0$ or $y = 0$. The initial configuration is $\mathcal{C}_0 = \langle \alpha_1, \alpha_2, \cdots, \alpha_m \rangle$.

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 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 $L(E_1) \cap L(E_2) \neq \phi$, 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. Let $\gamma = \mathcal{C}_0 \Longrightarrow \mathcal{C}_1 \Longrightarrow \cdots \Longrightarrow \mathcal{C}_k$ be an halting computation. Let us denote by $bin(\gamma)$ the string $b_1 b_2 \cdots b_k$ where $b_i \in \{0, 1\}$ and $b_i = 1$ iff the output neuron of the system Π sends a spike into the environment in the step i of γ, $b_i = 0$ iff it sends an anti-spike, and $b_i = \lambda$ if the step i generated no output. We denote by B the binary

alphabet $\{0, 1\}$ and by $COM(\Pi)$, the set of all halting computations of Π. Moreover, we define the language generated by Π by $L(\Pi) = \{bin(\gamma)|\gamma \in COM(\Pi)\}$.

When both the input and output neurons are considered, the system can be used as transducer. In each step of computation, each input neuron takes the input from the environment, while the output neuron produces the output to the environment. The input and output are spikes (representing binary digit 1) or anti-spikes (representing binary digit 0). We want to emphasize that no rule of the form $\overline{a}^c \to \overline{a}$ is used here.

3. Simulating Logic Gates

In this section we simulate the logic gates using SN P systems with anti-spikes in transducing mode.

Lemma 1. *Boolean AND and OR gates can be simulated by SN PA systems with three neurons in two steps.*

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

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 a spike, accumulating a total of three spikes and fires using a rule $a^3 \to \overline{a}$ sending an anti-spike (0) to the environment. But if we introduce a spike into each of the input neurons, the output neuron gets two anti-spikes and gets annihilated with a spike already present in it, remains with an anti-spike and fires using a rule $\overline{a} \to a$ producing a spike.

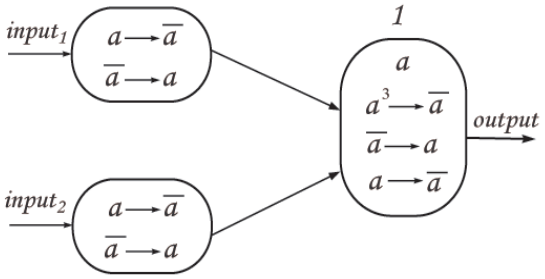

Fig. 1. SN P system with anti-spikes simulating AND gate.

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 one spike and fires using the rule $a \to \overline{a}$ sending an anti-spike to the environment. We can observe that it is simulating the AND gate correctly.

If we replace the rule $a \to \overline{a}$ with $a \to a$ in the output neuron of the above system, we obtain the SN PA system for an OR gate. □

Lemma 2. *The Boolean NOT gate can be simulated by an SN PA system with two neurons in two steps.*

Proof. The SN P system with anti-spikes simulating the NOT gate is depicted in Fig. 2. For synchronisation with OR and AND gates we added an output neuron so that output is produced after two steps. (Otherwise, the simulation is very simple, we can implement the gate with only one neuron in one step.)

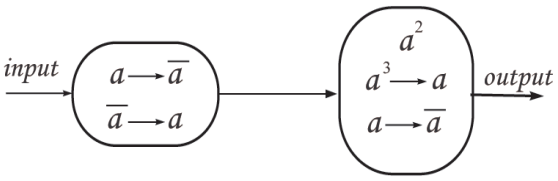

Fig. 2. SN P system with anti-spikes simulating NOT gate.

If an anti-spike is introduced, the output neuron will have three spikes in the next step and fires using the rule $a^3 \to a$, sending a spike. If a spike is introduced, it gets complemented in the input neuron and annihilates with a spike in the output neuron in the next step. So the output neuron has only one spike and produces an anti-spike using the rule $a \to \overline{a}$. Thus the NOT gate complements the input. □

4. Simulating Circuits

Here, we present the way to simulate any Boolean circuit using the AND, OR and NOT gates constructed in the previous section. But there is a need to construct synchronising module to ensure the synchronization among the gates.

Consider the following example $\neg(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$.
We use the SN P systems with anti-spikes for AND, OR and NOT gates. Let them be Π_{AND}, Π_{OR} and Π_{NOT}. The Boolean circuit corresponding to the above formula as well as the spiking system assigned to it are depicted in Fig. 3.

In order for the system that simulates the circuit to output the correct result it is necessary for each sub-system (that simulates the gates AND, OR and NOT) to receive the input from the above gate(s) at the same time. To this aim, we have to add a synchronizing SN P system Π_{SYN} as in Fig. 4. Generalizing the previous observations the following result holds:

Theorem 1. *Every Boolean circuit, whose underlying graph structure is a rooted tree, can be simulated by an SN PA system constructed from SN PA systems of types AND, OR and NOT by reproducing in the architecture of the SN PA system, the structure of the tree associated with the circuit.*

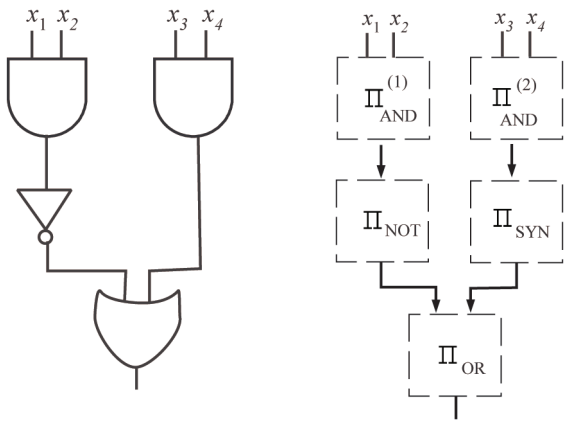

Fig. 3. Boolean circuit and corresponding SN P system with anti-spikes for $\neg(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$.

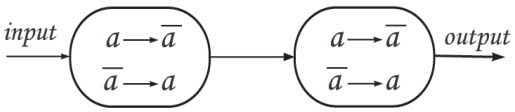

Fig. 4. Synchronizing SN P system with anti-spikes.

5. Computing with SN PA systems

As is shown below, SN PA systems can simulate in a direct manner several types of computing devices based on finite state transitions. The spike trains from the output neuron represent the language $L(\Pi)$ generated by the SN PA system Π. The following result is shown in [8].

Theorem 2. *Any regular language L over $\{0,1\}$ can be expressed as $L = L(\Pi)$ for some SN PA system Π.*

5.1. Simulating finite state transducers

Let $S = (Q, \Sigma, \Delta, \delta, \mu, q_1, F)$ be a deterministic finite state transducer with binary input and output, where $\Sigma = \Delta = \{0,1\}$, $Q = \{q_1, \cdots, q_n\}$, q_1 is the initial state, δ is the transition function that maps $Q \times \Sigma \to Q$ and μ is the output function from $Q \times \Sigma \to \Delta$.
We demonstrate that S can be simulated by an SN PA system.
Consider the following SN PA system:
Π_S=$(\{a, \overline{a}\}, \sigma_1, \sigma_2, \cdots, \sigma_{3n+1}, syn, 3n+1, 3n+1)$, with
$\sigma_i = (a, \{a \to a, a \to \overline{a}\}), i = 1, 2, \cdots, 3n,$
$\sigma_{3n+1} = (a^{3(n+1)}, \{a^{3(n+i)+1}/a^{3(n+i-j)+1} \to b' \mid \delta(q_i, 1) = (q_j, b)\} \cup$
$\{a^{3(n+i)-1}/a^{3(n+i-j)-1} \to b' \mid \delta(q_i, 0) = (q_j, b)\})$ where $b \in \{0,1\}$ and $b' = a$ if $b = 1$

and $b' = \overline{a}$ if $b = 0$, *syn* is the set of pairs $(i, 3n + i), (3n + i, i)$ with $1 \leq i \leq 3n$. The system is given in a pictorial way in Fig. 5. Note that n is the number of states, and that for each $1 \leq i \leq n$, q_i in Q is represented by $a^{3(n+i)}$. The number of spikes $a^{3(n+i)}$ in neuron σ_{3n+1} is referred to (or identified) as a state of Π_S. The manner of constructing Π_S is a modification of the one presented using extended SN P systems in [6].

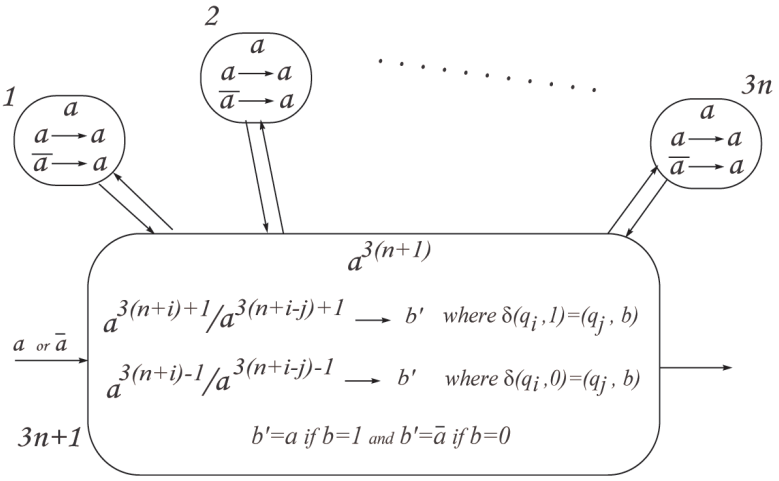

Fig. 5. An SN PA system simulating a transducer.

This system works as follows. Initially, the neuron σ_{3n+1} contains $3(n + 1)$ spikes which corresponds to the initial state q_1. Suppose that, in any step, neuron σ_{3n+1} contains $a^{3(n+i)}$(representing state q_i) and is ready to receive input a or $\overline{a}$ (representing 1 or 0 respectively) from environment. Depending on whether the input is a spike or anti-spike, neuron σ_{3n+1} can fire and emit a spike(if $b' = 1$) or anti-spike(if $b' = 0$) to environment by consuming $3(n + i - j) + 1$ or $3(n + i - j) - 1$ spikes leaving $3j$ spikes. It receives $3n$ spikes from neurons 1 to $3n$ accumulating a total of $3(n + j)$ spikes (representing q_j). Hence, one state transition $\delta(q_i, b) = (q_j, b')$ is simulated. This action is repeatedly performed in a number steps equal to the input length. When the system stops receiving the input, the neuron σ_{3n+1} will have a number of spikes which is a multiples of 3, hence the system halts. Thus, (with one step delay) for a given input $w = b_{i_1} b_{i_2} \cdots b_{i_r}$ in $\{0, 1\}^*$, the SN PA system Π_S produces an output $y = \mu(q_{i_1}, b_{i_1})\mu(q_{i_2}, b_{i_2}) \cdots \mu(q_{i_r}, b_{i_r})$ in $\{0, 1\}^*$, where the sequence of states: $z = q_{i_1} q_{i_2} \cdots q_{i_r}$ such that $\delta(q_{i_j}, b_{i_j}) = q_{i_{j+1}}$ for $j = 1, 2, \cdots, r - 1$ and $q_{i_1} = q_1$. We denote the output by $y = \Pi_S(w)$ and the sequence of states by $z = \Pi_{Sq}(w)$. A transducer S defines a function $w \to S(w)$, hence simulating S means that if $y = S(w)$, then $y = \Pi_S(w)$. Then it holds that y is generated by S (i.e., $\delta(q_1, w) \in F$) iff $z = \Pi_{Sq}(w)$ ends up with a final state in F (i.e., q_{i_r} is in F). We now define the language generated by Π_S as $\mathcal{N}(\Pi_S) = \{y \in \{0, 1\}^* \mid w \in \{0, 1\}^*, y = \Pi_S(w)$ and $\Pi_{Sq}(w)$ is in $Q^*F\}$.
Thus, the following theorem holds:

Theorem 3. *Any finite state transducer S can be simulated by some SN PA system* Π_S.

6. Solving SAT with SN PA systems

An instance of SAT is a Boolean formula in CNF $\gamma = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, i.e., a conjunction of clauses C_j, $1 \leq j \leq m$. Each clause is a disjunction of literals, i.e., occurrences of x_i or $\neg x_i$, built on the finite set $X = \{x_1, x_2, \cdots, x_n\}$ of Boolean variables. In what follows, we will require that no repetitions of the same literal may occur in any clause; in this way, a clause can be seen as a subset of all possible literals. An assignment of the variables $x_1, x_2, \cdots, x_n$ is a mapping $s : X \to \{0,1\}^n$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that Boolean formula γ is *satisfiable* if there exists an assignment of truth values to all the variables which occur in γ such that evaluation of γ gives 1 (true) as a result. The problem of SAT takes an arbitrary Boolean formula γ as input and asks if γ is satisfiable.

An SN PA system that solves the SAT problem in a non-deterministic uniform way is given in Fig. 6. The system has one module for each clause. As the construction is uniform, we code each clause C_j, $1 \leq j \leq m$, of the given instance of SAT as follows: 1 indicates the case when x_i appears in C_j, 0 indicates the case when $\neg x_i$ appears in C_j and λ (empty) indicates the absence of x_i in the clause C_j. That means that a spike, an anti-spike or no input (λ) are to be introduced in the input neurons of the system from the second step onwards and the output neuron emits a spike, if the given instance of SAT has a solution, otherwise sends an anti-spike.

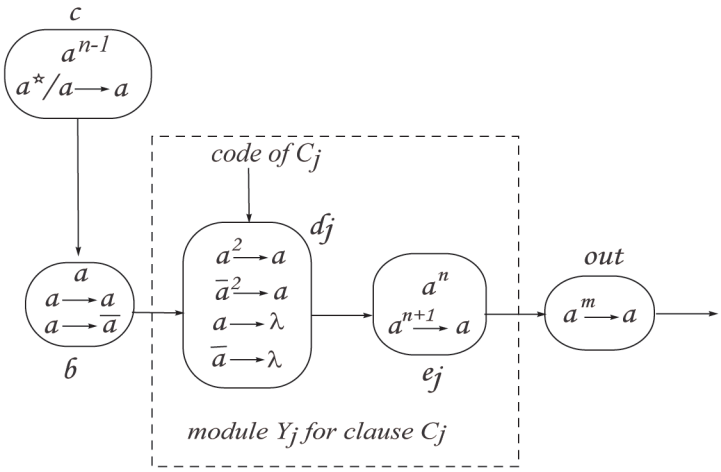

Fig. 6. An SN PA system solving SAT.

Actually, we consider m input neurons, one for each clause, and in each of them we introduce a sequence of n bits 1, 0 and λ (a spike, anti-spike or no input is sent inside

in the steps corresponding to the occurrence of 1,0 and λ respectively), describing the situation of each variable $x_1, \cdots, x_n$ with respect to the corresponding clause.

For instance, for the formula
$\gamma = (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3)$
we have two input neurons, the first one receiving the spike train $0\lambda 10$, and the second one receiving the spike train $\lambda 10\lambda$. Note the important fact that introducing the input takes only n steps. A module Y_j exists for each clause C_j, $1 \leq j \leq m$. Each module has a synapse going to the output neuron σ_{out}.

Neurons σ_c and σ_b are common to all modules; a synapse exists from σ_c to σ_b and from σ_b to all neurons σ_{d_j} of modules Y_j. Neuron σ_c provides a spike to neuron σ_b in each step for $n-1$ steps. The neuron σ_b non-deterministically produces a truth-assignment for the variables $x_1, \cdots, x_n$, using the choice between rules $a \to a$ and $a \to \overline{a}$. The spike needed for the truth assignment of x_1 is initially present in the neuron σ_b, while it gets the spike in each step for the next $n-1$ variables from the neuron σ_c.

An anti-spike coming out of σ_b is interpreted as the value *false* assigned to x_i, and a spike is interpreted as the value *true* assigned to x_i. Therefore, σ_{d_j} receives either an anti-spike or a spike from the neuron σ_b, and the spikes which codify the type of presence of x_i in clause C_j (no occurrence, negated, not negated).

In order to synchronize the checking performed in neurons σ_{d_j}, i.e., to bring here the truth assignment of variable x_i in the moment when the code of the presence of x_i in C_j arrives in this neuron, we use the neuron σ_c that supply spikes to neuron σ_b in each step for next $n-1$ steps. In each step beginning with the second step, all neurons σ_{d_j} receive both the truth assignment of x_i and the code of the way x_i is related with C_j.

As one can see from the previous explanations, in each step $2, 3, \cdots, n+1$, neurons σ_{d_j}, $1 \leq j \leq m$, receive a number of spikes/anti-spikes as follows:
1 anti-spike if x_i = false and x_i does not appear in C_j;
1 spike if x_i= true and x_i does not appear in C_j;
no spikes/anti-spikes if x_i= false and x_i appears in C_j;
2 spikes if x_i= true and x_i appears in C_j;
2 anti-spikes if x_i = false and $\neg x_i$ appears in C_j;
no spikes/anti-spikes if x_i= true and $\neg x_i$ appears in C_j

Thus, the rules of σ_{d_j} produce a spike only in the case when the clause C_j becomes true for the corresponding truth assignment of variable x_i. This spike reaches neurons σ_{e_j}. Each neuron σ_{e_j} has already n spikes and fires using the rule $a^{n+1} \to a$. The use of neuron e_j ensures the fact that σ_{out} receives at most one spike from each module Y_j, namely, only if clause C_j has been satisfied. All neurons σ_{e_j}, $1 \leq j \leq m$, are linked by a synapse to the output neuron σ_{out}. The neuron σ_{out} spikes (in step $n+3$) using a rule $a^m \to a$ only if the truth assignment produced non-deterministically by modules Y_j satisfies the formula γ.

It should be noted that the number of neurons of the system constructed above is $2m+3$, and that the computation takes a number of steps which is linear in n. Note that without anti-spikes [1] the solution requires double the number of steps and $3n^2+8m+5$ neurons.

7. Conclusion

In this paper, we have examined the computational efficiency of SN PA systems used as transducers. We show that the idea of encoding 1 as spike and 0 as anti-spike proves to be very efficient in simulating Boolean circuits, finite state transducers and solving NP-complete problems. We designed SN P systems simulating the operations of AND, OR and NOT gates. This motivates the modelling of CPU with SN P system with anti-spikes. We show that any instance of SAT in CNF, with n variables and m clauses is solved in a non-deterministic way with the number of neurons polynomial in m. Finally, investigating other computational complexity issues within this framework remains as a research topic of interest.

References

[1] Leporati A., Mauri G., Zandron C., Păun Gh., Pérez-Jiménez M. J., *Uniform Solutions to SAT and Subset-Sum by Spiking Neural P Systems*, Nat. Comput., **8** (4), pp. 681–702, 2008.

[2] Păun Gh., *Spiking Neural P Systems Used as Acceptors and Transducers*, CIAA, LNCS, **4783**, pp. 1–4, 2007.

[3] Ionescu M., Păun Gh., Yokomori T., *Spiking Neural P Systems*, Fundamenta Informaticae, **71** (2-3), pp. 279–308, 2006.

[4] Ionescu M., Sburlan D., *Some Applications of Spiking Neural P systems*, J. of Computing and Informatics, **27**, pp. 515–528, 2008.

[5] Ibarra O. H., Woodworth S., *Characterizations of Some Classes of Spiking Neural P Systems*, Nat. Comput., **7**, pp. 499–517, 2008.

[6] Ibarra O. H., Pérez-Jiménez M. J., Yokomori T., *On Spiking Neural P Systems*, Nat. Comput., **9** (2), pp. 475–491, 2009.

[7] Pan L., Păun Gh., *Spiking Neural P Systems with Anti-Spikes*, Int. J. of Computers, Communications and Control, **4** (3), pp. 273–282, 2009.

[8] Metta V. P., Krithivasan K., Garg D., *On String Languages Generated by Spiking Neural P Systems with Anti-Spikes*, Intern. J. Found. Computer Sci., **22** (1), pp. 15–27, 2011.