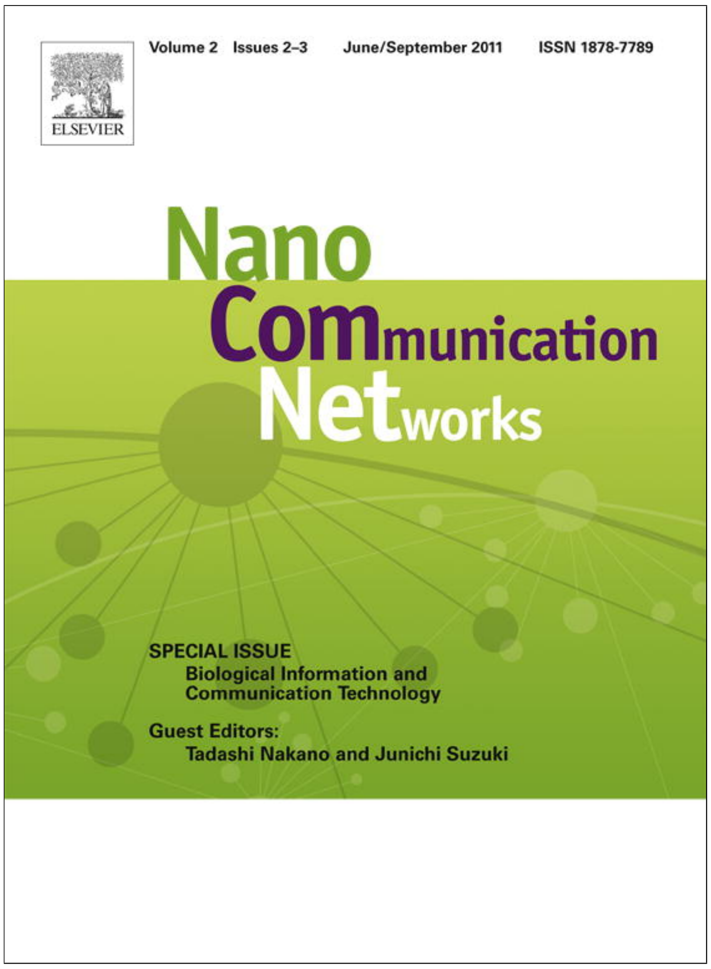

Volume 2 Issues 2–3 June/September 2011 ISSN 1878-7789
ELSEVIER
Nano Communication Networks
SPECIAL ISSUE
Biological Information and Communication Technology
Guest Editors:
Tadashi Nakano and Junichi Suzuki

Nano Communication Networks 2 (2011) 141–149

Contents lists available at SciVerse ScienceDirect

Nano Communication Networks

journal homepage: www.elsevier.com/locate/nanocomnet

# Modelling and analysis of spiking neural P systems with anti-spikes using Pnet lab

Venkata Padmavati Metta[a,*], Kamala Krithivasan[b], Deepak Garg[c]

[a] Bhilai Institute of Technology, Durg, India
[b] Indian Institute of Technology, Chennai, India
[c] Thapar University, Patiala, India

ARTICLE INFO

*Article history:*
Received 24 March 2011
Received in revised form 29 May 2011
Accepted 13 June 2011
Available online 24 June 2011

*Keywords:*
Spiking neural P system with anti-spikes
Petri net
Pnet Lab
Modelling

ABSTRACT

Petri Nets are promising methods for modelling and simulating biological systems. Spiking Neural P system with anti-spikes (SN PA systems) is a biologically inspired computing model that incorporates two types of objects called spikes and anti-spikes thus representing binary information in a natural way. In this paper, we propose a methodology to simulate SN PA systems using a Petri net tool called Pnet Lab. It provides a promising way for SN PA systems because of its parallel execution semantics and appropriateness to represent typical working processes of these systems. This enables us to verify system properties, system soundness and to simulate the dynamic behaviour.

## 1. Introduction

Spiking neural P systems (shortly called SN P systems), introduced in [3] as a variant of P systems [10], are mathematical models inspired by the neurobiological behaviour of neurons sending electrical pulses of identical voltages called spikes to neighbouring neurons.

An SN P system consists of a set of neurons placed in the nodes of a directed graph with each neuron having spiking and forgetting rules. The rules involve the spikes present in the neuron in the form of occurrences of a symbol $a$. The application of spiking rules transfer spikes to neighbouring neurons along the arcs of the graph (they are called synapses), whereas the forgetting rules simply forget some spikes present in the neuron.

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. An SN P system with anti-spikes (shortly called SN PA system) introduced in [7], 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). Each neuron in the system consists of an implicit annihilation rule of the form $a\bar{a} \rightarrow \lambda$; 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. So at any instant of time, a neuron in an SN P system with anti-spikes can have spikes or anti-spikes but not both.

The initial configuration of the system is described as initial number of spikes or anti-spikes present in each neuron. The SN PA system evolves in a synchronous fashion, meaning that a global clock is assumed and in each time unit all neurons work in parallel with each neuron which can use a rule should do it, but using only one rule at a time (sequential locally). Using the rules, we can define transitions among configurations. A sequence

* Corresponding author. Tel.: +91 7882358392; fax: +91 7882358392.
*E-mail addresses:* vmetta@gmail.com (V.P. Metta), kamala@iitm.ac.in (K. Krithivasan), deep108@yahoo.com (D. Garg).

1878-7789/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.nancom.2011.06.002

of transitions among configurations, starting from initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be used. With any computation whether halting or not together with output produced in such a case, yielding notions of functionality and computational power of SN PA systems including various aspects of computing.

It is extremely important to simulate these models to portray the system behaviour. Such models can shed insight into complex processes and suggest new directions for research. Scientists can study and analyse such models to make predictions about the behaviour of the system under different conditions and to discuss novel relationships among the different components of a system. The ability to predict system behaviour with a model helps to evaluate model completeness as well as improve our understanding of the system.

A modelling methodology that is especially tailored for representing and simulating concurrent dynamic systems is Petri Nets. An advantage of Petri nets is that they have a visual representation and simulation that facilitates user comprehension. Petri Net tools enable users to verify system properties, verify system soundness, and to simulate the dynamic behaviour.

Different variants of P systems are translated into Petri nets to complement the functional characterization of their behaviour. In [5,4], Petri nets with localities were introduced to represent some variants of membrane systems. In [8,9], SN P systems with delay and SN P systems with anti-spikes are translated into new variants of Petri net models. However, all these new variants of Petri nets typically lack the tools for building models, for executing and observing simulation experiments. In [1], a tool for simulating a simple and extended SN P system is introduced that yields only the transition diagram of a given system in a step-by-step mode and it lacks step-by-step graphical simulation of the system. Spiking neural P systems with anti-spikes are new variants of SN P systems. We studied the languages generated by the SN P systems with anti-spikes in [6] and proved that some of the languages that cannot be generated using standard SN P systems can be generated using SN P systems with anti-spikes but no tools are available to simulate these variants.

This paper introduces the direct translation of standard SN PA systems into Petri net models that can be simulated using existing Petri net tools. As the procedure is direct, it involves less complexity in translation and also using the notions and tools already developed for Petri nets, one can describe the internal process occurring during a computation in the SN PA system in a graphical way. Perhaps the greatest advantages of Petri nets are a solid mathematical foundation and the large number of techniques being developed for their analysis. These include reachability analysis, invariant analysis (a technique using linear algebra), transformations (including reductions) preserving desired properties, structure theory and formal language theory. We considered Pnet Lab—a Java based simulation tool for Petri nets to analyse the SN P systems. Pnet Lab allows the parallel firing of all enabled transitions after resolving the conflicts that can efficiently simulate the parallel use of rules in all neurons in each step. It also allows the user defined guard function that can encode the regular expression associated with each rule. Last but not least, Pnet Lab has the advantage that it is extremely light-weight and being implemented in Java it is platform independent.

This paper is organized as follows. We start with Section 2 by giving brief introduction about SN PA system. In Section 3, we discuss the Petri net model considered for translations. Section 4 gives a brief introduction about Pnet Lab. Using these definitions as basis in Section 5, we translate an SN PA system into an equivalent Petri net model that can be simulated using Pnet Lab. Section 6 gives analysis results for the SN PA system considered in Section 2 through Pnet Lab.

### 1.1. Notation

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

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

Let the alphabet $\Sigma$ be the set $\{a_1, \dots, a_n\}$. The letter distribution, $\phi(w)$, of a $\Sigma$-word $w$ is the $n$-tuple $\langle N_1, \dots, N_n\rangle$ with $N_i$ the number of occurrences of $a_i$ in $w$. The Parikh set, $\phi(L)$, of a $\Sigma$-language $L$ is $\{\phi(w) | w \in L\}$.

A language $L \subseteq \Sigma^*$ is said to be regular if there is a regular expression $E$ over $\Sigma$ such that $L(E) = L$. The regular expressions are defined using the following rules. (i) $\phi$, $\lambda$ and each $a \in \Sigma$ are regular expressions. (ii) if $E_1, E_2$ are regular expressions over $\Sigma$, then $E_1 + E_2$, $E_1E_2$ and $E_1^*$ are regular expressions over $\Sigma$, and (iii) nothing else is a regular expression over $\Sigma$. 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 $\left\{\epsilon, a, a^2, a^3, \dots\right\}$. The positive closure $a^+ = a^* - \{\lambda\}$. If $\Sigma$ is a singleton then the Parikh set of the language denoted by regular expression $E$ over $\Sigma$, $L(E)$ is $\{|w| \mid w \in L(E)\}$.

A multiset over a set $X$ is a function $m : X \to N = \{0, 1, 2, \dots\}$. Multiset $m$ is empty if there are no $x$ such that $x \in m$ by which we mean that $x \in X$ and $m(x) \geq 1$. The cardinality of $m$ is $|m| = \Sigma_{x\in X} m(x)$. For two multisets $m$ and $m'$ over $X$, the sum $m + m'$ is the multiset given by the formula $(m + m')(x) = m(x) + m'(x)$ for all $x \in X$, and if $k \in N$ then $k.m$ is the multiset given by $(k.m)(x) = k.m(x)$ for all $x \in X$. We denote $m' \leq m$ whenever $m'(x) \leq m(x)$ for all $x \in X$, and if $m' \leq m$, then the difference $m - m'$ is $m(x) - m'(x)$ for all $x \in X$.

A multiset over $X$ may be represented as a string of elements from $X$; for example, $a\overline{a}aa\overline{a} = a^3\overline{a}^2$ (three occurrences of $a$s and two occurrences of $\overline{a}$) denotes the multiset $m$ over $X = \{a, \overline{a}\}$ such that $m(a) = 3$, and $m(\overline{a}) = 2$. The empty string (multiset) will be denoted by $\lambda$.

## 2. Spiking neural P system with anti-spikes

First we recall the definition of the SN P system with anti-spikes (or SN PA system).

**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

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

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

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

   (a) $n_i$ is the *multiset of spikes or anti-spikes* contained by the neuron.
   (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 $\overline{a}$, while $b, b' \in \{a, \overline{a}\}$, and $r \geq 1$.
      ii. $b^r \to \lambda$, for some $r \geq 1$, with the restriction that $b^r \notin L(E)$ for any rule $E/b^r \to b'$ of type (1) from $R_i$;
3. syn $\subseteq \{1, 2, 3, \dots, m\} \times \{1, 2, 3, \dots, m\}$ with $(i, i) \notin$ syn for $1 \leq i \leq m$ (*synapses* among neurons);
4. $i_0 \in \{1, 2, 3, \dots, m\}$ 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 $nb$'s such that $b^n \in L(E)$ and $n \geq r$. When neuron $\sigma_i$ sends a $b$, it is replicated in such a way that one $b'$ is sent to all neurons $\sigma_j$ such that $(i, j) \in$ syn. The rules of type $b^r \to \lambda$ are forgetting rules; $r$ number of $b's$ are simply removed ("forgotten") when applying. As 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.

There is an additional fact that $a$ and $\overline{a}$ cannot stay together, so 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 that annihilation rule has priority over spiking and forgetting rules, so the neurons always contain either only spikes or anti-spikes.

lhs$(v)$ and rhs$(v)$ give the multiset of spikes/anti-spikes present in the left- and right-hand sides of rule $v$ respectively. As 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 $E = b^r$ then we will write it in the simplified form $b^r \to b'$.

The standard SN P system with delay works in a similar way as that of the SN PA system but deals with only one type of object called *spike* $(a)$ and so there exists no annihilation rules. The spiking rules have a delay associated with them and are of the form $E/a^r \longrightarrow a; t$. When neuron $\sigma_i$ uses the rule, its spike is replicated in such a way that one spike is sent to all neurons $\sigma_j$ such that $(i, j) \in$ syn, and $\sigma_j$ is open at that moment. If $t = 0$, then the spikes are emitted immediately, if $t = 1$, then the spikes are emitted in the next step and so on. In the case $t \geq 1$, if the rule is used in step $d$, then in step $d, d + 1, d + 2, \dots, d + t - 1$, the neuron is closed and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost, biology calls this the refractory period). In step $t + d$, the neuron spikes and becomes open again, hence can receive spikes (which can be used in step $t+d+1$). If a neuron $\sigma_i$ fires and either it has no outgoing synapse, or all neurons $\sigma_j$ such that $(i, j) \in$ syn are closed, then the spike of neuron $\sigma_i$ is lost; the firing is allowed, it takes place, but results in no new spikes. In an SN P system without delay all the rules have zero delay. Since in this paper we always deal with systems without delay, the delay $(t = 0)$ is never specified in the rules.

**Definition 2.2** (*Configuration*)**.** The configuration of the system is described $\mathcal{C} = \langle n_1, n_2, \dots, n_m \rangle$ where $n_i$ is the multiset written in the form $n_i = a^x\overline{a}^y$, where $x$ is the number of spikes and $y$ is the number of anti-spikes present in neuron $\sigma_i$. Because a neuron always contains spikes or anti-spikes, either $n_i(a) = 0$ or $n_i(\overline{a}) = 0$.

A global clock is assumed in the 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. 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 $\Pi$ such that $V(i) = r_{ij}$, $r_{ij}$ is a spiking or forgetting rule from $R_i$, i.e. $|V(i)| = 0$ or 1 where $1 \leq i \leq m$. If no rule is applicable from $\sigma_i$ then $V(i) = r_{i0}$. If a vector rule $V$ is enabled at a configuration $\mathcal{C} = \langle n_1, n_2, \dots, n_m \rangle$ then $\mathcal{C}$ can evolve to $\mathcal{C}' = \left\langle n'_1, n'_2, \dots, n'_m \right\rangle$ (after applying annihilation rules in each neuron in an exhaustive way), where

$$n'_i = n_i - \text{lhs}(V(i)) + \sum_{(j,i)\in\text{syn}} \text{rhs}(V(j)).$$

**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 $\mathcal{C}$ and $\mathcal{C}'$ of $\Pi$ we denote by $\mathcal{C} \overset{V}{\Longrightarrow} \mathcal{C}'$, if there is a direct transition from $\mathcal{C}$ to $\mathcal{C}'$ in $\Pi$.

A computation of $\Pi$ 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 $\mathcal{C}$ is non-deterministic in the sense that there may be different vector rules applicable to $\mathcal{C}$, 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 [11].

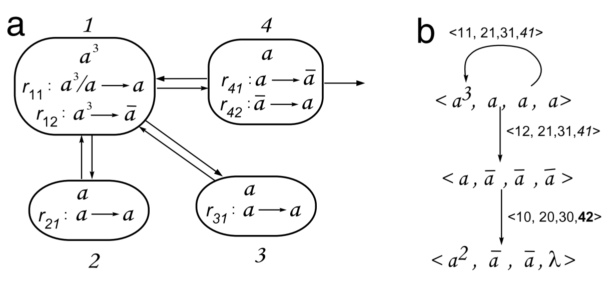


**Fig. 1.** SN P system with anti-spikes.

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

$$\Pi = (\{a, \bar{a}\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \text{syn}, 4), \quad \text{with}$$
$$\sigma_1 = (a^3, \{a^3/a \to a, a^3 \to \bar{a}\}),$$
$$\sigma_2 = (a, \{a \to a\}),$$
$$\sigma_3 = (a, \{a \to a\}),$$
$$\sigma_4 = (a, \{a \to \bar{a}, \bar{a} \to a\}),$$
$$syn = \{(1, 2), (2, 1), (1, 4), (4, 1), (1, 3), (3, 1)\}.$$

We have four neurons, with labels 1, 2, 3, 4; neuron 4 is the output neuron. Initially, neuron 1 has three spikes with non-determinism between its first two rules and neurons 2, 3 and 4 have one spike each. The initial configuration of the system is $\langle a^3, a, a, a\rangle$.

The evolution of the system $\Pi$ can be analysed on a transition diagram as that from Fig. 1(b) because the number of configurations reachable from the initial configuration is finite, we can place them in the nodes of a graph and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Fig. 1(b), we have also indicated the rules used in each neuron with the following conventions; for each $r_{ij}$ we have written only the subscript $ij$; when a neuron $i = 1, 2, 3, 4$ uses no rule, we have written $i0$.

The functioning can easily be followed on this diagram, so that we only briefly describe it. We start with spikes in all neurons. Neuron 1 can behave non-deterministically choosing one of the two rules. As long as neuron 1 uses the rule $a^3/a \to a$, the computation cycles in the initial configuration sending a spike to neurons 2, 3 and 4; neuron 4 uses its first rule and sends an anti-spike to environment and neuron 1. Neurons 2 and 3 use their rules and send a spike to neuron 1. So neuron 1 receives one anti-spike and two spikes (and two spikes are already present in it), after using the annihilation rule, the neuron will have again three spikes. Neurons 2, 3 and 4 will have one spike each.

If neuron 1 uses its second rule $a^3 \to \bar{a}$, the three spikes are consumed and an anti-spike is sent to other three neurons. So neuron 1 will have one spike and neurons 2, 3 and 4 will have one anti-spike each, reaching the configuration $\langle a, \bar{a}, \bar{a}, \bar{a}\rangle$. In the next step neurons 1, 2 and 3 cannot fire and neuron 4 uses the rule $\bar{a} \to a$ sending a spike to environment and neuron 1, reaching the configuration $\langle a^2, \bar{a}, \bar{a}, \lambda\rangle$ and the system halts.

## 3. Petri net

A Petri [13] net is a bipartite graph with two kinds of nodes, place nodes are represented with circles having tokens and transition nodes are represented with bars or boxes. The directed arcs connecting places to transitions and transitions to places may be labelled with an integer weight, but if unlabelled are assumed to have a weight equal to 1. Now we introduce the class of Petri nets with transitions having guards, to be used in the translation.

**Definition 3.1** (*Petri Net*)**.** A Petri net with guard is represented by $\mathcal{N} = (P, T, A, W, G, M_0)$, where

$P = \{P_1, P_2, P_3, \ldots, P_m\}$ is a finite, non-empty set of places.
$T = \{T_1, T_2, T_3, \ldots, T_n\}$ is a finite, non-empty set of transitions.
$A \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs which connect places with transitions and transitions with places.
$W : A \longrightarrow N$ assigns weight $W(f)$ to elements of $f \in A$ denoting the multiplicity of unary arcs between the connecting nodes.
$G : T \longrightarrow \{true, false\}$, the guard function maps each transition $T_i$ to Boolean expression, which specifies an additional constraint which must be fulfilled before the transition is enabled.

The initial marking $M_0 = \{n_1, n_2, \ldots, n_m\} \in P$, each $n_i$ is the number of tokens initially associated with each place $P_i$ and $m$ is the number of places in the net $\mathcal{N}$.

A place $P_i$ is an input (or an output) place of a transition $T_j$ iff there exists an arc $(P_i, T_j)$ (or $(T_j, P_i)$ respectively) in set A. The sets of all input and output places of a transition $T_j$ are denoted by $I(T_j) = \{P_i : (P_i, T_j) \in A\}$ and $O(T_j) = \{P_i : (T_j, P_i) \in A\}$ respectively. Similarly, the sets of input and output transitions of a place $P_i$ are denoted by $I(P_i) = \{T_j : (T_j, P_i) \in A\}$ and $O(P_i) = \{T_j : (P_i, T_j) \in A\}$ respectively. A place without any output transition is called output place. Output place only receives tokens but does not send any tokens to other places.

**Definition 3.2** (*Marking*)**.** A marking (state) assigns to each place $P_i$ a non-negative integer $k$, we say that place $P_i$ is marked with $k$ tokens. Pictorially we place $k$ black dots (tokens) in place $P_i$. A marking is denoted by $M$, an $m$-vector where $m$ is the total number of places. *Sub marking* of a Petri net is the marking of some of its places.

The state or marking of Petri net is changed by the occurrence of transition. When a transition is enabled, it may be fired to remove a number of tokens from each input place equal to the weight of the connecting input arc and create a number of new tokens at each output place equal to the weight of the connecting output arc.

Firing rules in the Petri net model are:

1. Transition $T_j$ is enabled iff $T_j$ satisfies the guard condition and its every input place has at least as many tokens as the weight of the input arcs,

$$M(P_i) \geq W(P_i, T_j) \quad \forall P_i \in I(T_j)$$
$$G(T_i) = \text{true}.$$

2. Upon firing the transition $T_j$ removes number of tokens from each of its input places equal to the weight of the input arcs and deposits number of tokens into the output places equal to the weight of output arcs.

Concurrency is also a concept that Petri net systems represent in an extremely natural way. Two transitions are concurrent at a given marking if they can be fired at the same time, i.e. simultaneously. The set of all transitions enabled by a marking $M$ is denoted by $E(M)$. When a transition fires, a token is removed from each of its input places and a token is added to each of its output places. This determines a new marking in a net, a new set of enabled transitions, and so on. An important concept in Petri nets is that of conflict. Conflict occurs between transitions that are enabled by the same marking, where the firing of one transition disables the other. A major feature of net is that they do not define in any way how and when a given conflict should be resolved, leading to non-determinism on its behaviour.

**Definition 3.3** (*Step*)**.** A step is a set $U$ of transitions which fires at a marking $M$ after resolving conflicts and is denoted by $M[U\rangle$. The input and output places of step $U$ are given by

$$\text{IN}_{\mathcal{N}}U(p) = \sum_{t \in U} W(p,t) \quad \text{and}$$
$$\text{OUT}_{\mathcal{N}}U(p) = \sum_{t \in U} W(t,p) \quad \text{for each } p \in P.$$

A step $U$ which is enabled at a marking $M$ can be executed leading to the marking $M' = M + \text{OUT}_{\mathcal{N}}U(p) - \text{IN}_{\mathcal{N}}U(p)$. We denote this by $M\,[U\rangle\,M'$. A step $U$ is a maximal step at a marking $M$ if $M\,[U\rangle$ and there is no transition $t'$ such that $M\left[U + \{t'\}\right\rangle$ and for every place $p \in P$, transition $t \in U$, $t$ can only be executed if it satisfies the guard function.

A Petri net system $\mathcal{N}$ with maximal concurrency is such that for each markings $M$ and $M'$ if there is a step $U$ such that $M\,[U\rangle\,M'$, then $U$ is a maximal step. In this paper we are considering only maximal concurrency semantics of the Petri nets.

A computation of a Petri net $\mathcal{N}$ is a finite or infinite sequences of executions starting from the initial marking and every marking appearing in such a sequence is called reachable. A major strength of Petri nets is their support for the analysis of many properties and problems associated with concurrent systems such as reachability, boundedness and liveness. The firing of an enabled transition will change the token distribution in a net according to the transition. A sequence of firings will result in a sequence of markings.

*Coverability tree* is a tree representation of all possible markings with initial marking as the root node and nodes as the markings reachable from $M_0$ and arcs represent the transition firing. A *reachability graph* is a graph where each node represents a Petri net marking, with arcs connecting each marking with all of its next markings. The reachability graph defines a net's state space (i.e. the set of reachable states). Reachability is a fundamental basis for studying the dynamic properties of any system. A marking $M_n$ is reachable from initial marking $M_0$ if a sequence of firings that transforms $M_0$ to $M_n$. The reachability problem for Petri net is the problem of finding if a marking $M_i$ is reachable from the initial marking $M_0$. Formally, a Petri net with a given marking is said to be in deadlock if and only if no transition is enabled in the marking. A Petri net where no deadlock can occur starting from a given marking is said to be live. A place-invariant ($P$-invariant) is a subset of places whose total number of tokens remains unchanged under any execution of the system. A transition-invariant ($T$-invariant) is a multiset of transitions whose execution in a certain order will leave the distribution of tokens unchanged. Generally, Petri nets are analysed using tools to study important behavioural properties of the system like invariants, reachability, liveness, boundedness.

## 4. Pnet Lab

Pnet Lab tool provides interactive simulation, analysis and supervision for Petri nets. It allows modelling and analysis of coloured Petri Nets, place-transition nets, timed/untimed. We can build an arc (guard) function by combining the built in functions or using several mathematical functions in accordance with the C/C++ syntax. This paper makes use of built in function *ntoken*$(i)$ that returns the number of tokens present in the place $p_i$. We can also write user defined guard functions. The DFA for the regular expression $E$ is translated into a user defined guard function that enables a transition when the number of spikes (in the form of sequence of $a$'s) present in the neuron is in $L(E)$. It also allows the firing of multiple transitions in a single step and resolves conflicts.

Pnet Lab manages conflicts by using the following resolution policies:

1. Predefined Scheduling order: PNet Lab assigns a static priority to the transition in conflict, based on the order in which they have been drawn;
2. Same firing rate: transition in conflict relation have the same firing probability;
3. Stochastic firing rate: transition in conflict relation has a firing probability defined a priori by the user.

For Petri net models, the computation of $T$–$P$-invariant, minimal siphons and traps, pre-incidence, post-incidence and incidence matrices and coverability tree is available. A detailed manual about Pnet Lab can be found in [12]. This paper makes use of built in functions of Pnet Lab.

## 5. SN PA system to Petri net

In this section, we propose a formal method to translate SN PA systems into Petri nets suitable for simulation using Pnet Lab.

Three places are used to represent each neuron. The marking of the places $P_{3i-2}$ and $P_{3i-1}$ gives the number of spikes and anti-spikes present in the neuron $\sigma_i$ respectively. The place $P_{3i}$ is added to allow at most one transition to fire from each input place corresponding to $\sigma_i$. $P_{3m+1}$ and $P_{3m+2}$ are the places corresponding to the environment and shows the number of spikes and anti-spikes sent out by the output neuron. Every spiking and forgetting rule is one-to-one mapped to a transition in

$T$. Every annihilation rule in $\sigma_i$ is represented with two transitions $T_{ia}$ and $T_{ib}$. $T_{ia}$ is fired if the number of spikes is more than the number of anti-spikes, otherwise $T_{ib}$ is fired. Regular expressions are translated into guard functions which further control the firing of transitions.

In an SN PA system, the annihilation rule is applied in each neuron after the application of spiking rules. To simulate this behaviour of SN PA system, a place $P_{3m+3}$ with no tokens is introduced and all transitions that corresponds to spiking rules are executed if place $P_{3m+3}$ has no tokens. This is implemented in Pnet Lab by adding a guard function to each of these transitions that checks the number of tokens in place $P_{3m+3}$ and are fired if $ntoken(P_{3m+3}) = 0$. The outgoing arc of each of these transitions are connected to the place $P_{3m+3}$. The annihilation rules are applied if any spiking rules are applied in the previous step so a guard function $ntoken(P_{3m+3}) > 0$ is added to each transition corresponding to the annihilation rule. The place $P_{3m+3}$ is connected to the transition $T_0$ which fires with the annihilated transitions, clearing its contents and thus allowing spiking transitions to fire in the next step.

**Definition 5.1** (*SN PA System to Petri Net*). Let $\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \dots, \sigma_m, \text{syn}, i_0)$ be an SN P system, then the corresponding Petri net $\mathcal{NL}_\Pi \overset{df}{=} (P, T, A, W, G, M_0)$, where

1. $P \overset{df}{=} \{P_1, P_2, \dots, P_{3m}, P_{3m+1}, P_{3m+2}, P_{3m+3}\}$ is the set of places.
2. $T \overset{df}{=} T_1 \cup T_2 \cup \cdots T_m \cup T_0$ where each group of transitions $T_i$, $1 \le i \le m$ contains a distinct transition $T_{ik}$ for every rule of $r_{ik} \in R_i$. For each annihilation rule that is internally present in neuron $\sigma_i$, $T_i$ contains two transitions $T_{ia}$ and $T_{ib}$.
3. add $(P_{3i-2}, T_{ia})$, $(P_{3i-1}, T_{ia})$, $(P_{3i}, T_{ia})$ and $(T_{ia}, P_{3i})$ to $A$ with
   $W(P_{3i-2}, T_{ia}) = ntoken(P_{3i-1})$,
   $W(P_{3i-1}, T_{ia}) = ntoken(P_{3i-1})$,
   $W(P_{3i}, T_{ia}) = 1$ and $W(T_{ia}, P_{3i}) = 1$
   $G(T_{ia}) \overset{df}{=}$ (if $ntoken(P_{3i-2}) \ge ntoken(P_{3i-1})$ and $ntoken(P_{3m+3}) > 0$ then return *true* else return *false*)
   The execution of $T_{ia}$ consumes all tokens from its input places and leaves $ntoken(P_{3i-2}) - ntoken(P_{3i-1})$ tokens in place $P_{3i-2}$.
4. add $(P_{3i-2}, T_{ib})$, $(P_{3i-1}, T_{ib})$, $(P_{3i}, T_{ib})$ and $(T_{ib}, P_{3i})$ to $A$ with
   $W(P_{3i-2}, T_{ib}) = ntoken(P_{3i-2})$,
   $W(P_{3i-1}, T_{ib}) = ntoken(P_{3i-2})$,
   $W(P_{3i}, T_{ib}) = 1$ and $W(T_{ib}, P_{3i}) = 1$
   $G(T_{ib}) \overset{df}{=}$ (if $ntoken(P_{3i-1}) > ntoken(P_{3i-2})$ and $ntoken(P_{3m+3}) > 0$ then return *true* else return *false*)
5. add $(i_0, m+1)$ to *syn*. This adds an arc from output neuron to the environment.

```
for every place $(i, j) \in$ syn do
for every transition $T_{ik} \in T_i$ do
add $(P_{3i}, T_{ik})$, $(T_{ik}, P_{3i})$ and $(T_{ik}, P_{3m+3})$ to $A$ with arc weight 1.
if $r_{ik}$ is of the form $E/a^k \to b$ where $b = a$ or $\overline{a}$ or $\lambda$ then
$G(T_{ik}) \overset{df}{=}$ (if $ntoken(P_{3i-2}) \in$ Parikh set of $L(E)$ and $ntoken(P_{3m+3}) = 0$ then return true else return false)
add $(P_{3i-2}, T_{ik})$ to $A$ with $W(P_{3i-2}, T_{ik}) = k$
if $b = a$ then
add $(T_{ik}, P_{3j-2})$ to $A$ with $W(T_{ik}, P_{3j-2}) = 1$
else if $b = \overline{a}$ then
add $(T_{ik}, P_{3j-1})$ to $A$ with $W(T_{ik}, P_{3j-1}) = 1$
end if
else if $r_{ik}$ is of the form $E/\overline{a}^k \to b'$ where $b' = a$ or $\lambda$ then
$G(T_{ik}) \overset{df}{=}$ (if ntoken$(P_{3i-1}) \in$ Parikh set of $L(E)$ and ntoken$(P_{3m+3}) = 0$ then return true else return false)
add $(P_{3i-1}, T_{ik})$ to $A$ with $W(P_{3i-1}, T_{ik}) = k$
if $b' = a$ then add $(T_{ik}, P_{3j-2})$ to $A$ and set $W(T_{ik}, P_{3j-2}) = 1$
end if
end if
end for
end for
add $(P_{3m+3}, T_0)$ to $A$ with arc weight as $ntoken(P_{3m+3})$.
```

6. *for* $i = 1$ to $m$, set
   $M_0(P_{3i-2}) \overset{df}{=} n_i(a)$
   $M_0(P_{3i-1}) \overset{df}{=} n_i(\overline{a})$
   $M_0(P_{3i}) \overset{df}{=} 1$.

To capture the very tight correspondence between the SN PA system $\Pi$ and Petri nets $\mathcal{NL}_\Pi$, we introduce a straightforward bijection between configurations of $\Pi$ and markings of $\mathcal{NL}_\Pi$, based on the correspondence between places and neurons.

Let $\mathcal{C} = \langle n_1, n_2, \dots, n_m \rangle$ be a configuration of an SN PA system $\Pi$. Then the corresponding sub marking $\phi(\mathcal{C})$ of $\mathcal{NL}_\Pi$ is given by $\phi(\mathcal{C})(P_{3i-2}) \overset{df}{=} n_i(a)$ and $\phi(\mathcal{C})(P_{3i-1}) \overset{df}{=} n_i(\overline{a})$ for every place where $1 \le i \le m$ of $\mathcal{NL}_\Pi$.

Similarly, for any vector rule $V = (r_{1j_1}, r_{2j_2}, \dots, r_{mj_m})$ of $\Pi$, we define an enabled step $\xi(V)$ of transitions of $\mathcal{NL}_\Pi$ such that $\xi(V)(T_{ij}) \overset{df}{=} r_{ij}$ for every $T_{ij} \in T$ and $j \ne a$ or $b$. It is clear that $\phi$ is a bijection from the configurations of $\Pi$ to the markings of $\mathcal{NL}_\Pi$, and that $\xi$ is a bijection from vector rules of $\Pi$ to enabled steps of $\mathcal{NL}_\Pi$.

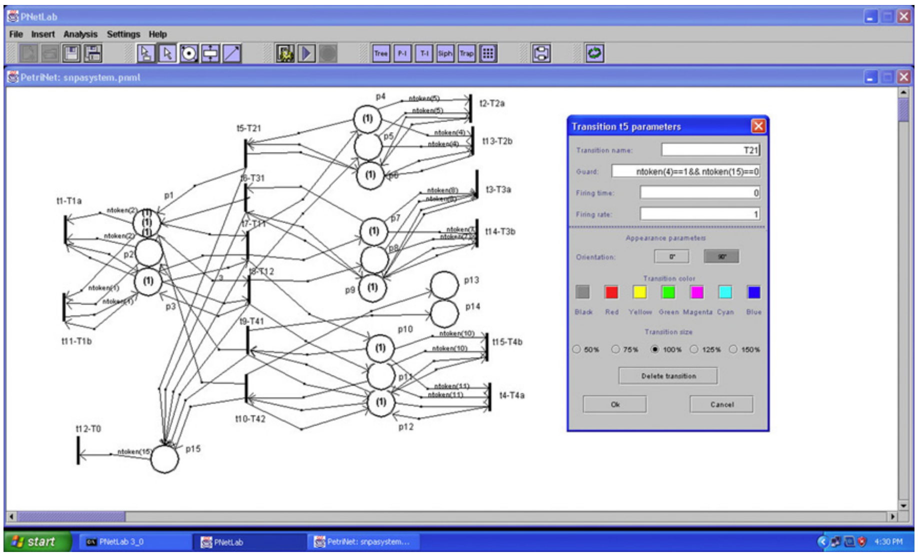
Now we can formulate a fundamental property concerning the relationship between the dynamics of the SN PA system $\Pi$ and that of the corresponding Petri net:

$$\mathcal{C} \xRightarrow{V} \mathcal{C}' \quad \text{if and only if} \quad \phi(\mathcal{C})[\xi(V)\rangle[H\rangle\phi(\mathcal{C}')$$

where $H$ is an immediate step and may contain $T_{ia}$ or $T_{ib} \in T$ for every $1 \le i \le m$ and transition $T_0$.

Since the initial configuration of $\Pi$ corresponds through $\phi$ to the initial sub marking of $\mathcal{NL}_\Pi$, the above immediately implies that the computations of $\Pi$ coincide with the locally sequential and globally maximal concurrency semantics of the net $\mathcal{NL}_\Pi$.

It can be observed that the structure of neurons in $\Pi$ is used in the definitions of the structure of the net $\mathcal{NL}_\Pi$ (i.e., in the definitions of places, transitions and the weight function). Let $\mathcal{C}$ be a configuration of $\Pi$ and there is a vector rule $V$ enabled at $\mathcal{C}$ reaching a configuration $\mathcal{C}'$. As there is a mapping between configuration and markings, $\phi(\mathcal{C})$ is marking of net $\mathcal{NL}_\Pi$ corresponding to the configuration $\mathcal{C}$ of $\Pi$. There is a one-to-one mapping between the rules in the SN PA system and transitions in net. So there exists a step $[\xi(V)\rangle$ enabled at the marking $\phi(\mathcal{C})$. After the execution of the steps $[\xi(V)\rangle$ and $H$, the system reaches the configuration $\phi(\mathcal{C}')$. We can prove only if part in the similar way. So the evolution of the Petri net $\mathcal{NL}_\Pi$ is same as the evolution of the SN PA system $\Pi$.



Transition name: T21

Guard: ntoken(4)==1&& ntoken(15)==0

Firing time: 0

Firing rate: 1

Appearance parameters

Orientation: 0° [✓ 90°]

Transition color

Black Red Yellow Green Magenta Cyan Blue

Transition size

50% 75% [✓ 100%] 125% 150%

Delete transition

Ok Cancel

**Fig. 2.** Petri net model for SN PA system in Fig. 1.

## 6. Simulation with Pnet Lab

We explain the simulation with an example. Fig. 2 shows the Petri net model for the SN P system in Example 2.1 modelled using Pnet Lab. Each transition is named as $tl - Tik$, where $tl$ is the transition name given by the tool and $Tik$ is the transition name given as per methodology discussed in the previous section. We can also find the invariants. In [2], it is proved that finding invariants enables us to establish the soundness and completeness of the system. The tool also outputs the coverability tree which is not shown in the figure.

Fig. 3 gives the output of the step-by-step simulation of the model in Pnet Lab. $p_1$ and $p_2$ are places corresponding to neuron $\sigma_1$ for storing spikes and anti-spikes respectively. The symbol $\{1\}$ in the marking column indicates the presence of a token in that place. so the combined marking of the places $p_1$ and $p_2$ gives the configuration of the first neuron. If we consider the sub marking i.e. $(p_1, p_2)$ for neuron 1, $(p_4, p_5)$ for neuron 2, $(p_7, p_8)$ for neuron 3 and $(p_{10}, p_{11})$ for neuron 4, the initial marking is $\langle(3, 0), (1, 0), (1, 0), (1, 0)\rangle$ which is similar to the initial configuration of the SN PA system in Fig. 1. At this marking, after the firing of transitions $t5 - T21, t6 - T31, t8 - T12, t9 - T41$ (corresponding to rules 21,31,12,41 of $\Pi$), the system reaches the next sub marking $\langle(2, 1), (0, 1), (0, 1), (0, 1)\rangle$. As the number of tokens in place $p15$ is greater than zero, the transitions $t1 - T1a, t12 - T0$, corresponding to the annihilation rules will be fired in the next step again reaching the same configuration as that of the SN P system, i.e. $\langle(1, 0), (0, 1), (0, 1), (0, 1)\rangle$ (shown in pass.2 of Fig. 3). The transition enabled at this marking is $t10$–$T42$ followed by the transitions corresponding to annihilation rules $t12$–$T0$ reaching the final marking $\langle(2, 0), (0, 1), (0, 1), (0, 0)\rangle$. We can observe from Fig. 3 that the configurations reachable from initial configuration of the SN P system are same as the sub markings reachable in the corresponding Petri net model from the initial sub marking. So we conclude that the Petri net model in Fig. 2 accurately simulates the working of the SN P system $\Pi$.

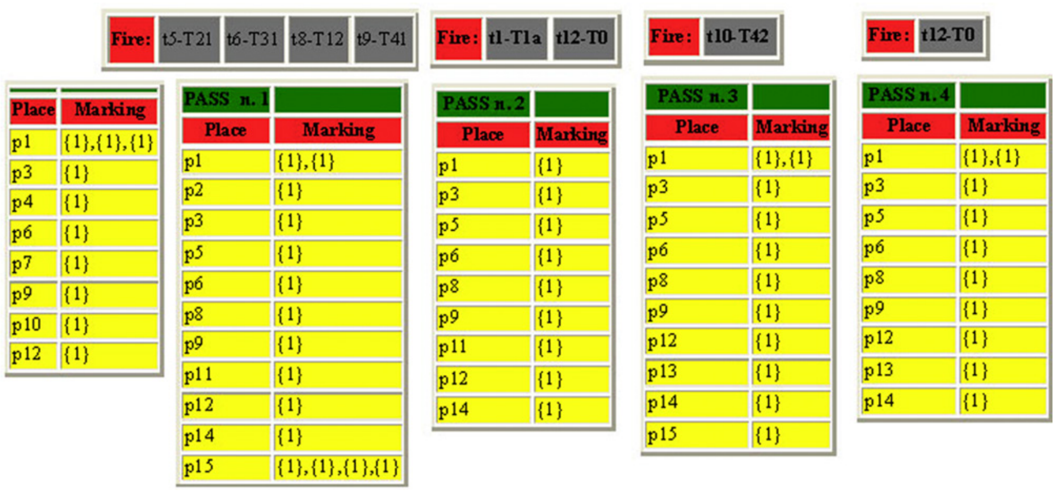
### 6.1. The behavioural properties of SN PA systems derived from Petri nets

Many useful behavioural properties such as reachability, boundedness, liveness of Petri nets have been investigated. We also introduce these properties for SN PA systems.

For an SN PA system, we define structural analysis which can identify properties that are conserved during execution of the modelled system. It may provide insights to the system. Such properties include the following:

1. *Boundedness*: Checking that there is no infinite accumulation of tokens in a place. This may correspond to an accumulation of spikes at a particular neuron.
2. *T-Invariants*: Identifying a set of transitions that have to fire from some initial marking to return the Petri net to that marking. $T$-invariants indicate the presence of cycles that are in a state of continuous operation.
3. *Reachability*: Deciding whether a certain marking (state) is reachable from another marking. This type of analysis can be used to determine whether certain outcomes are possible, given a modelled net and an initial marking (initial state), or to determine whether certain configurations are reachable when specific rules are inhibited.
4. *Terminating*: the sequence of transitions between configurations of a given SN P system is finite, i.e., the computation of the SN PA system always halts.
5. *Deadlock-free*: each reachable configuration enables a next step.
6. *Liveness*: it is deadlock-free and there is a sequence containing steps.

**Theorem 1.** *If the Petri net for a given SN PA system $\Pi$ is terminating, then the SN PA system $\Pi$ is terminating.*



| Place | Marking |
|---|---|
| p1 | {1},{1},{1} |
| p3 | {1} |
| p4 | {1} |
| p6 | {1} |
| p7 | {1} |
| p9 | {1} |
| p10 | {1} |
| p12 | {1} |

Fire: t5-T21 t6-T31 t8-T12 t9-T41

| PASS n. 1 | |
|---|---|
| Place | Marking |
| p1 | {1},{1} |
| p2 | {1} |
| p3 | {1} |
| p5 | {1} |
| p6 | {1} |
| p8 | {1} |
| p9 | {1} |
| p11 | {1} |
| p12 | {1} |
| p14 | {1} |
| p15 | {1},{1},{1},{1} |

Fire: t1-T1a t12-T0

| PASS n. 2 | |
|---|---|
| Place | Marking |
| p1 | {1} |
| p3 | {1} |
| p5 | {1} |
| p6 | {1} |
| p8 | {1} |
| p9 | {1} |
| p11 | {1} |
| p12 | {1} |
| p14 | {1} |

Fire: t10-T42

| PASS n. 3 | |
|---|---|
| Place | Marking |
| p1 | {1},{1} |
| p3 | {1} |
| p5 | {1} |
| p6 | {1} |
| p8 | {1} |
| p9 | {1} |
| p12 | {1} |
| p13 | {1} |
| p14 | {1} |
| p15 | {1} |

Fire: t12-T0

| PASS n. 4 | |
|---|---|
| Place | Marking |
| p1 | {1},{1} |
| p3 | {1} |
| p5 | {1} |
| p6 | {1} |
| p8 | {1} |
| p9 | {1} |
| p12 | {1} |
| p13 | {1} |
| p14 | {1} |

**Fig. 3.** Report of markings in the steps of simulation in Pnet Lab.

**Proof.** If the SN PA system is not terminating, according to the definition of termination for SN PA systems, there exists an infinite step sequence. When the SN PA system is encoded by the Petri net, there also exists an infinite step sequence. Every step is one-to-one mapped to a transition in the Petri net, so the sequence of transition in the Petri net is not finite. Thus, this Petri net is not terminating. □

**Theorem 2.** *If the Petri net for a given SN P system $\Pi$ is deadlock-free, then the SN P system $\Pi$ is deadlock-free.*

**Theorem 3.** *If the Petri net for a given SN PA system $\Pi$ has liveness, then the SN PA system $\Pi$ has liveness.*

**Theorem 4.** *If the Petri net for a given SN PA system $\Pi$ is bounded, then the SN PA system $\Pi$ is bounded.*

**Proof.** The proofs of Theorems 2–4, are the same as that for Theorem 1. □

## Conclusion

SN PA systems are biologically inspired computing models that involve the use of two types of objects called spikes and anti-spikes and thus model the systems working with binary data in a very natural way. A formalism to study these models and validating them is needed. The Petri net tool called Pnet lab allows the parallel execution of transitions. It enables us to model the globally parallel firing semantics of all SN PA systems. They also allow the definition of functions on arcs and transitions. With numerous functionalities available with Pnet lab, we succeeded in modelling the entire work of SN PA systems. At present the algorithms only enable the simulation of different variants of SN P systems without delay. We are therefore interested in developing a method of validation and verification for SN P systems with delay.

## Acknowledgements

We are grateful to Marian Gheorghe for many inspiring discussions. We would also like to thank the referees for their helpful comments.

## References

[1] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, D. Ramírez-Martínez, A software tool for verification of spiking neural P systems, Natural Computing 7 (4) (2008) 485–497.
[2] O.H. Ibarra, M.J. Pérez-Jiménez, T. Yokomori, On spiking neural P systems, Natural Computing 9 (2) (2010) 475–491.
[3] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundamenta Informaticae 71 (2006) 279–308.
[4] J. Kleijn, M. Koutny, A Petri net model for membrane system with dynamic structure, Natural Computing 8 (4) (2009) 781–796.
[5] J. Kleijn, M. Koutny, G. Rozenberg, Process semantics for membrane system, Journal of Automata, Languages and Combinatorics 11 (2006) 321–340.
[6] K. Krithivasan, V.P. Metta, D. Garg, On string languages generated by spiking neural P systems with anti spikes, International Journal of Foundations of Computer Science 22 (1) (2011) 15–27.
[7] P. Linqiang, Gh. Păun, Spiking neural P systems with anti-spikes, International Journal of Computers, Communications & Control 4 (2009) 273–282.
[8] V.P. Metta, K. Krithivasan, D. Garg, Modeling spiking neural P systems using timed Petri nets, NaBIC, IEEE Xplore, 2009, doi:10.1109/NABIC.2009.5393490.
[9] V.P. Metta, K. Krithivasan, D. Garg, Representation of spiking neural P systems with anti-spikes through Petri nets, in: the Proceedings of BIONETICS, LNICST, Springer (2010) (forthcoming).
[10] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (2000) 108–143.
[11] Gh. Păun, Spiking neural P systems used as acceptors and transducers, in: CIAA, in: LNCS, vol. 4783, Springer, 2007, pp. 1–4.
[12] Pnet Lab: a Petri net tool. http://www.automatica.unisa.it/PnetLab.html.
[13] W. Reisig, G. Rozenberg, Lectures on Petri nets, in: LNCS, vols. 1491, 1492, Springer-Verlag, Berlin, 1998.

**Venkata Padmavati Metta** received her MCA in Computer Applications from MANIT, Bhopal and currently pursuing her Ph.D. in Computer Science and Engineering at Thapar University. She has 10 years teaching experience and is an Associate Professor at Bhilai Institute of Technology, India. Her main research fields are formal language theory, algorithms and membrane computing.

**Kamala Krithivasan** received her Ph.D. from the University of Madras, and she joined the Indian Institute of Technology Madras (IITM) in 1975. With more than 30 years of teaching and research experience at IITM, she is currently Professor at the Department of Computer Science and Engineering, in which she served as Chairperson during 1992–1995. Her research interests include formal language theory and unconventional models of computing like DNA computing, membrane computing and discrete tomography. She has published a large number of research papers, lectured at many universities and gave numerous invited talks at recognized international conferences. A recipient of the Fulbright fellowship in 1986, Professor Kamala is also a fellow of the Indian National Academy of Engineering.

**Deepak Garg** has done his Ph.D. in the area of efficient algorithm design. He has 11 years teaching experience and is currently Professor at Thapar University, India. His active research interests are designing efficient algorithms, bioinformatics and knowledge management. He started his career as a Software Engineer in IBM Corporation Southbury, USA.