Simulation of Spiking Neural P Systems using Pnet Lab

Venkata Padmavati Metta
Bhilai Institute of Technology, Durg
vmetta@gmail.com

Kamala Krithivasan
Indian Institute of Technology, Madras
kamala@iitm.ac.in

Deepak Garg
Thapar University, Patiala
deep108@yahoo.com

Abstract: In this paper we propose a methodology for translating Spiking Neural P systems without delay into Petri net models. This enables us to study several aspects concerning the formal verification of SN P systems, by using results and tools regarding various classes of Petri nets. A tool called Pnet Lab is used to simulate and check certain properties of these systems.

Key words: Spiking Neural P System, Petri net, Pnet Lab.

1 Introduction

P systems (membrane systems) were introduced in [10] as parallel computing devices inspired by the hierarchical structure of membranes in living organisms, and the biological processes which take place in and between cells. Spiking Neural P systems (SN P systems) [4] are a class of P systems which use ideas from neural computing, more precisely the way neurons communicate with each other by sending electrical signals of identical voltage, called spikes, through synapses - links established with neighbouring neurons. The spikes of neurons look alike, but the timing and number of spikes entering a neuron determines the way the information is encoded.

SN P system is pictorially described as a directed graph where nodes represent the 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 time unit. The configuration of the system at any time is represented by the number of spikes present in each neuron at that time. Configuration is changed by the locally sequential and globally maximal application of rules, such a step is called a transition. Thus one can define computations (sequences of transitions from the initial configuration) which can then be interpreted as specifying

a result of the operation of the SN P system. Typically, one defines a notion of successful (or halting) computation together with the output produced in such case, by considering one or more neurons as output neurons, yielding to notions of functionality and computational power of the SN P system [11], including various aspects of complexity.

For efficient formalization, to deal with the implementation and formal correctness of SN P systems, this paper proposes a formal method based on Petri nets. Petri nets play an important role in the modelling, analysis and verification of concurrent systems. Petri nets [12] are graphical as well as algebraic modelling tools, applicable to a large variety of information processing systems. They consist of places and transitions. Places represent the objects which indicate the availability of resources, represented by tokens. Thus places are used to represent neurons and spikes are indicated with tokens. Transitions are actions which can occur depending upon the availability of resources and thus represent spiking and forgetting rules inside the neurons. It is worth noting that as far as the rules are concerned, SN P systems are highly concurrent in nature and Petri nets are successful modelling paradigm, which allows concurrent systems to be described in a formal yet graphical and well readable way. It is attractive to adopt Petri nets to model SN P systems. So the rich theoretical concepts and practical tools from well-developed Petri nets could be introduced in the current research of SN P systems.

There are different interactions between classes of membrane systems and Petri nets that have been investigated so far. In [5] it is provided an overview of all these investigations. In this paper we are interested in those interactions investigating the role of Petri nets as a tool to express behavioural semantics for membrane systems. This has been discussed in [5] and thoroughly investigated in [7, 6] Petri nets with localities have been introduced to represent some variants of membrane systems. In [1] some problems like producer-consumer are expressed in both formalisms in order to analyse their modelling capabilities. 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 [2], a tool for simulating 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 of step-by-step simulation of the system.

This paper introduces the direct translation of standard SN P systems without delay 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 P system in a graphical way. Perhaps the greatest advantages of Petri nets are a solid mathe-

matical foundation and the large number of techniques being developed for their analysis. These include: reachability analysis, invariants 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 organised as follows. We start with section 2 by giving brief introduction about standard SN P system. In section 3, we discuss the Petri net model considered for translations. Using these definitions as basis in section 4, we translate an SN P system into an equivalent Petri net model that can be simulated using Pnet Lab. The section 5 gives analysis results for SN P systems through Pnet Lab.

1.1 Notation

We recall here few definitions and notations related to 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 string 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) 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_1 E_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

Here we are introducing the basic class of SN P systems.

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

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

1. $\mathrm{O} = \{a\}$ is the singleton alphabet (a is called *spike*) ;
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons, of the form

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

   where

   (a) $n_i \geq 0$ is the *initial number of spikes* contained by the cell;

   (b) R_i is a finite set of *rules* of the following two forms:

      i. $E/a^r \longrightarrow a;t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$.
      The number of spikes present in the neuron is described by the regular expression E, r spikes are consumed and it produces a spike, which will be sent to other neurons after t time units

      ii. $a^s \longrightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin \mathrm{L}(E)$ for any rule $E/a^r \longrightarrow a;t$ 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 $E/a^r \longrightarrow a;t$ are spiking rules, and they are possible only if the neuron contains n spikes such that $a^n \in \mathrm{L}(E)$ and $n \geq r$. If $E = \phi$ then the rule is applied only if the neuron contains exactly r spikes. When neuron σ_i spikes, its spike is replicated in such a way that one spike is sent to all neurons σ_j such that $(i, j) \in syn$, and σ_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 $d, d+1, d+2, ..., d+t-1$, 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 σ_i fires and either it has no outgoing synapse, or all neurons σ_j such that $(i, j) \in syn$ are closed, then the spike of neuron σ_i is lost; the firing is allowed, it takes place, but results in no new spikes.

The rules of type $a^s \longrightarrow \lambda$ are forgetting rules; s 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, $a^s \longrightarrow \lambda$ is applied only if the neuron contains exactly s spikes. In this paper we will

consider systems without delays (i.e., $t = 0$ in all rules). Because all rules have the delay 0, we write them in the simpler form $E/a^c \longrightarrow a$, hence omitting the indication of the delay.

Definition 2.2 *(Configuration)* The configuration of the system is described by the number of 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 $n_i \geq 0$ spikes. The initial configuration of the system is described by $C_0 = \langle n_1, n_2, n_3, \ldots, n_m \rangle$, where n_i is the number of spikes present in i^{th} neuron.

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. 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 $V(i) = r_{ij}$, r_{ij} is a spiking or forgetting rule from R_i i.e $\mid V(i) \mid = 0$ or 1 where $1 \leq i \leq m$. If no rule is applicable from σ_i then $V(i) = r_{i0}$. 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 $\mathcal{C}' = \langle n'_1, n'_2, \ldots, n'_m \rangle$, where

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

where $lhs(V(i))$ and $rhs(V(i))$ gives the number of spikes present in the left and right hand sides of rule $V(i)$ respectively.

Definition 2.3 *(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.

The evolution of the system Π can be analysed on a transition diagram as that from Fig.1(b), because the system is finite, the number of configurations reachable from the initial configuration is finite too, hence, 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.

A computation halts if it reaches a configuration where no rule can be used. With any computation halting or not we associate a spike train, a sequence of digits of 0 and 1, with 1 appearing in position which indicates the steps when the output neuron sends spikes out of the system. One of the neurons is considered to be the output neuron, and its spikes are sent to the environment. With any spike train we can associate various numbers which are considered as computed

by the system.

Example 2.1 Consider the graphical representation of an SN P system in Fig.1, 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. Fig.1(a) represents the initial configuration of a non-deterministic SN P system Π. It is formally represented as:

$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn, 3)$, with

$\sigma_1 = (2, \{a^2/a \longrightarrow a, a^2 \longrightarrow a, a \longrightarrow \lambda\})$,

$\sigma_2 = (1, \{a \longrightarrow a\})$,

$\sigma_3 = (3, \{a^3 \longrightarrow a, a^2 \longrightarrow \lambda, a \longrightarrow a\})$,

$syn = \{(1,2),(2,1),(1,3),(2,3)\}$.

This SN P system works as follows. We have three neurons, with labels 1, 2 and 3; neuron 3 is the output neuron. Initially neuron 2 has one spike with a rule and neuron 1 has two spikes with three rules and non-determinism between its first two rules. So the initial configuration of the system is $\mathcal{C}_0 =< 2, 1, 3 >$. All

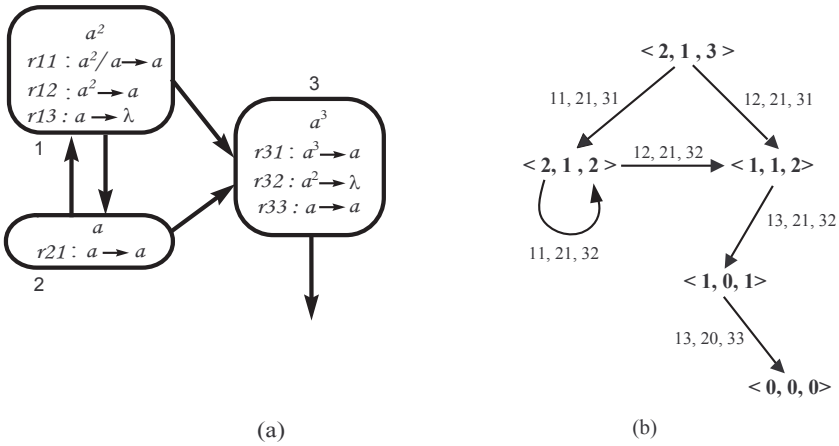

Figure 1: Spiking neural P system and its transition diagram

neurons can fire in the first step, with neuron 1 choosing non-deterministically

between its first two rules. Output neuron 3 sends its spike to the environment by using $a^3 \longrightarrow a$. If the neuron 1 uses the rule $a^2/a \longrightarrow a$, the system reaches the configuration $< 2, 1, 2 >$. The two spikes in output neuron 3 are forgotten in the next step. Neurons 1 and 2 also exchange their spikes; thus, as long as neuron 1 uses the rule $a^2/a \longrightarrow a$, it retains one spike and receives a spike from neuron 2, thus reaches the same configuration again.
At any instance of time, starting from step 1, neuron 1 can choose its second rule $a^2 \longrightarrow a$, which consumes its two spikes and sends a spike to neuron 2 and 3 reaching the configuration $< 1, 1, 2 >$. In the next step neuron 1 forgets its spike will have one spike, reaching the configuration $< 1, 0, 1 >$. In the last step the neuron 3 outputs a spike and the systems halts.
The evolution of the system Π can be analysed on a transition diagram as that from Fig.1(b), because the system is finite, the number of configurations reachable from the initial configuration is finite too, hence, 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. The rules used in each neuron are indicated with the following conventions; for each r_{ij} only the subscript ij used and with $i0$ when a neuron i uses no rule.

3 Petri net

A Petri 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 $M_0 = \{n_1, n_2, \cdots n_m\} \in P$, 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 the 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,
   $\mathrm{M}(P_i) \geq \mathrm{W}(P_i, T_j) \qquad \forall\ P_i \in I(T_j)$
   $\mathrm{G}(T_i)$=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 Petri nets 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 mark-

ing M after resolving conflicts and is denoted by $M\,[U\rangle$. The input and output places of step U are given by

$IN_{\mathcal{N}}U(p)= \sum_{t\in U} W(p,t)$ and
$OUT_{\mathcal{N}}U(p)= \sum_{t\in U} W(t,p)$ for each $p \in P$

A step U which is enabled at a marking M can be executed leading to the marking $M' = M + OUT_{\mathcal{N}}U(p)$ - $IN_{\mathcal{N}}U(p)$. 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 $M\,[U + \{t'\}\rangle$ 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 sequence 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 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 etc.

4 SN P System to Petri Net

In this section, we propose a formal method to translate SN P systems without delay into Petri nets suitable for simulation using Pnet Lab.

Every neuron in the SN P system is one-to-one mapped to a place in P. Every rule is one-to-one mapped to a transition in T. Regular expressions are translated into guard functions that further control the transitions.

The bindings of transitions are found by matching incoming arc expressions with tokens marking input places and checking guard satisfaction. To describe locally sequential semantics of the SN P system, a synchronizing place is added to each place to allow at most one transition (one rule) to fire from each input place.

Definition 5.1 *(SN P system without delay to Petri net)* Let Π=(O, σ_1, σ_2, ..., σ_m, *syn*, i_0) be an SN P system, then the corresponding Petri net $\mathcal{NL}_\Pi \stackrel{df}{=} (P, T, A, W, G, M_0)$, where

1. $P \stackrel{df}{=} \{P_1, P_2, \cdots, P_m, P_{m+1}, P_{11}, P_{21}, \cdots, P_{m1}\}$ is the set of places. P_i, $1 \leq i \leq m$ are places corresponding to the neurons σ_i and P_{i1} is the synchronizing place for place P_i and P_{m+1} is the place corresponding to environment.

2. $T \stackrel{df}{=} T_1 \cup T_2 \cup \cdots T_m$ where each group of transitions T_i contains a distinct transition T_{ik} for every rule of $r_{ik} \in R_i$.
   for every place $P_i \in P$, $1 \leq i \leq m$ do
   for every transition $t = T_{ik} \in T_i$ do
   $G(t) \stackrel{df}{=}$ (if $a^{n_i} \in L(E)$ return *true* else return $false$) where $n_i = M(P_i)$
   $W(P_i, t) = lhs(r_{ik})$, $W(P_{i1}, t) = 1$
   if $i = m$ *then* $W(t, P_{m+1}) = rhs(r_{ik})$
   else for every $(i, j) \in syn$, $W(t, P_j) = rhs(r_{ij})$
   end if
   $W(t, P_{i1}) = 1$
   end for
   end for

3. *for* every place $P_i \in P$, its initial marking is $M_0(P_i) \stackrel{df}{=} n_i$.

To capture the very tight correspondence between the SN P system without delay Π and Petri nets $\mathcal{NL}_\Pi$, we introduce a straight forward bijection between configurations of Π and markings of $\mathcal{NL}_\Pi$, based on the correspondence between places and neurons.

Let $\mathcal{C} =< n_1, n_2, \cdots, n_m >$ be a configuration of an SN P system Π. Then the corresponding sub marking $\phi(\mathcal{C})$ of $\mathcal{NL}_\Pi$ is given by $\phi(\mathcal{C})(P_i) \stackrel{df}{=} n_i$ for every place P_i where $1 \leq i \leq m$ of $\mathcal{NL}_\Pi$.

Similarly, for any vector rule $V = (r_{1j_1}, r_{2j_2}, \cdots, r_{mj_m})$ of Π, we define a step $\xi(V)$ of transitions of $\mathcal{NL}_\Pi$ such that $\xi(V)(T_{ij}) \stackrel{df}{=} r_{ij}$ for every $T_{ij} \in T$. It is clear that ϕ is a bijection from the configurations of Π to the markings of $\mathcal{NL}_\Pi$, and that ξ is a bijection from vector rules of Π to steps of $\mathcal{NL}_\Pi$.
We now can formulate a fundamental property concerning the relationship between the dynamics of the SN P system Π and that of the corresponding Petri net:

$\mathcal{C} \stackrel{V}{\Longrightarrow} \mathcal{C}'$ *if and only if* $\phi(\mathcal{C})[\xi(V) > \phi(\mathcal{C}')$.

Since the initial configuration of Π corresponds through ϕ to the initial sub marking of $\mathcal{NL}_\Pi$, the above immediately implies that the computations of Π coincide with the concurrency semantics of the Petri net $\mathcal{NL}_\Pi$.
The reader might by now have observed that the structure of neurons in Π is

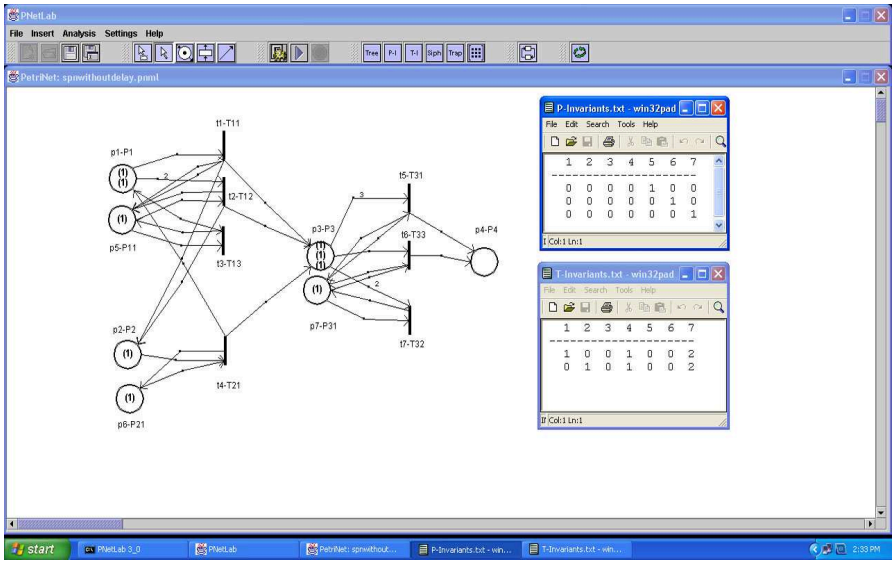

Figure 2: Petri net model for SN P system in Example1 and the invariants

used in the definitions of the structure of the Petri net $\mathcal{NL}_\Pi$ (i.e., in the definitions of places, transitions and the weight function). Let $\mathcal{C}$ be a configuration of Π and there is a vector rule V enabled at $\mathcal{C}$ reaching a configuration $\mathcal{C}'$. As there is a mapping between configuration and sub markings, $\phi(\mathcal{C})$ is the sub marking of Petri net $\mathcal{NL}_\Pi$ corresponding to the configuration $\mathcal{C}$ of Π. There is a one-to-one mapping between the rules in the SN P system and transitions in Petri net.

For locally sequential and globally maximal concurrency firing semantics of SN P systems, the synchronizing places P_{i1} for each place P_i are created to allow at most one transition to fire from each input place. So there exists a step $\xi(V)$ enabled at the sub marking $\phi(\mathcal{C})$. After the execution of the step 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 the same as the evolution of the SN P system Π.

5 Simulation with Pnet Lab

Pnet Lab is a software for Petri nets that provides interactive simulation, analysis and supervision. It allows modelling and analysis of Colored Petri Nets, place-transition nets, timed/untimed. 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. 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 have a firing probability defined a priori by the user;

A detailed manual about Pnet Lab can be found in [13].
Fig.2 shows the Petri net model for the SN P system in Example. 1 modelled using Pnet Lab. Each place is named as $pi-Pj$, where pi is the place name given by the tool and Pj is the place name given as per methodology discussed in the previous section . The transitions are also named in the same manner. Fig.2 also gives the invariants. In [3] it is proved that finding invariants enable 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. The symbol $\{1\}$ in the marking column indicates the presence of a token in that place. If we consider the sub marking for first three places (the places corresponding to the neurons), the initial marking is $< 2, 1, 3 >$ which is similar as that of the SN P system in Fig.1. At this marking, after the firing of transitions $t1 -$

$T11, t4-T21, t5-T31$ (corresponding to rules 11,21,31 of Π), the system reaches the next sub marking $< 2, 1, 2 >$. It stays in the same marking as long as $t1 - T11, t4 - T21, t7 - T32$ are fired. When it chooses the transition $t2 - T12$ instead of $t1 - T11$, we reach $< 1, 1, 2 >$. We can observe from the Fig.3 that the configurations reachable from initial configuration of the SN P system are same as the marking reachable in the corresponding Petri net model from the initial marking. So we conclude that the Petri net model in Fig.2 accurately simulates the working of the SN P system Π.

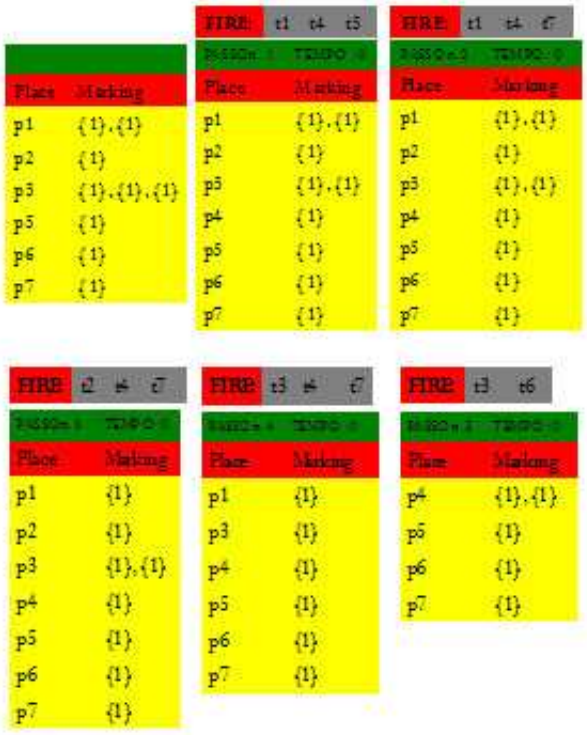

Figure 3: Report of markings in the steps of simulation

5.1 The Behavioural Properties of SN P 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 P systems.
For a SN P system, we define

1. **Terminating**: the sequence of transitions between configurations of a given SN P system is finite, i.e., the computation of the SN P system always halts.
2. **Deadlock-free**: each reachable configuration enables a next step.
3. **Liveness**: it is deadlock-free and there is a sequence containing steps.
4. **Boundedness**: An SN P system is said to be k-bounded or simply bounded if the number of copies of objects in each neuron for every reachable configuration will not exceed a finite number k.

Theorem 1. *If the Petri net for a given SN P system Π is terminating, then the SN P system Π is terminating.*

Proof. If the SN P system is not terminating, according to the definition of termination for SN P systems, there exists an infinite step sequence. When the SN P 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 Π is deadlock-free, then the SN P system Π is deadlock-free.*

Theorem 3. *If the Petri net for a given SN P system Π has liveness, then the SN P system Π has liveness.*

Theorem 4. *If the Petri net for a given SN P system Π is bounded, then the SN P system Π is bounded.*

Proof. The proofs of Theorem 2, 3, 4, are the same as for Theorem 1.

Conclusion

This paper showed how the problem of analysing SN P systems without delay can be approached by using Petri net tool. We discussed a methodology for translating SN P system into Petri net model that can be simulated using Pnet Lab - a simulation tool for Petri nets is used to study the properties like coverability tree, P-invariants and T-invariants. Our future work involves the simulation of more variants of spiking neural P systems using Petri nets.

References

1. F. Bernardini, M. Gheorghe, M. Margenstern, S. Verlan, "Producer/Consumer in Membrane Systems and Petri nets", Ed.Berlin: Springer-Verlag, LNCS 4497, 43-52, 2007.
2. 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), 485-497, 2008.
3. O. H. Ibarra, M.J. Pérez-Jiménez,T. Yokomori, "On Spiking Neural P systems" Natural Computing 9(2), 475-491, 2010.
4. M. Ionescu, Gh. Păun, T. Yokomori, "Spiking Neural P Systems", Fundamenta Informaticae 71, 279-308, 2006.
5. J. Kleijn, M. Koutny, "A Petri net Model for Membrane system with Dynamic Structure", Natural Computing 8(4), 781-796, 2009.
6. J. Kleijn, M. Koutny, "Petri nets and Membrane computing". In "The Oxford Handbook of Membrane Computing" (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 389-412, 2010.
7. J. Kleijn, M. Koutny, G. Rozenberg, "Process Semantics for Membrane System", Journal of Automata, Languages and Combinatorics 11, 321-340, 2006.
8. V. P. Metta, K. Krithivasan, D. Garg, "Modeling Spiking Neural P systems using Timed Petri nets", NaBIC, IEEE xplore, DOI: 10.1109/NABIC.2009.5393490, 2009.
9. V. P. Metta, K. Krithivasan, D. Garg, "Representation of Spiking Neural P Systems with Anti-spikes through Petri nets" forthcoming in the Proceedings of BIONETICS, LNICST, Springer, 2010.
10. Gh. Păun, "Computing with Membranes", Journal of Computer and System Sciences 61, 108-143, 2000.
11. Gh. Păun, "Spiking Neural P Systems used as Acceptors and Transducers", CIAA, LNCS 4783, 1-4, Springer, 2007.
12. W. Reisig, G. Rozenberg G, "Lectures on Petri nets", Lecture notes in computer science 1491, 1492 Springer-Verlag, Berlin, 1998.
13. Pnet Lab: A Petri net tool, *http://www.automatica.unisa.it/PnetLab.html*.