Protocol Modeling in Spiking Neural P systems and Petri nets

Venkata Padmavati Metta
Bhilai Institute of Technology

Kamala Krithivasan
Indian Institute of Technology

Deepak Garg
Thapar University

ABSTRACT

In this paper we present the relation between Spiking Neural P (SN P) systems and Petri nets by focusing on modeling simplex stop-and-wait protocol. The SN P system for the protocol is constructed and also translated it into equivalent Petri net with a corresponding semantics. It is then observed a direct correspondence between the Petri net representation of the proposed model and standard solution based on Petri nets already present in the literature.

Key words: Petri net, Spiking neural P system, simplex stop-and-wait protocol, modeling.

1. INTRODUCTION

Membrane computing is an emerging Branch of natural computing which deals with distributed and parallel computing devices of a bio-inspired type, which are called membrane systems, or P systems [9]. Spiking neural P systems are variants of membrane systems devised based on the observation that neurons send electrical impulses (also called spikes) along the axons to other neurons. It is a new model in the area of neural computation.

An SN P system [4] consists of a set of neurons (membranes) connected by synapses. The structure is represented as a directed graph where the directed edges represent the synapses and the nodes represent the neurons. The system has only a single unit of information referred to as spike and is represented by symbol *a*. The spikes are stored in the neurons. The rules are assigned to neurons that allow for sending information to other neurons in the form of electrical impulses (also called spikes), which are summed up at the target cell; the application of the rules depends on the contents of the neuron (which, in general case, is described by regular sets). Within each time unit, the system is transformed by the rules which are applied in concurrent fashion. The system is synchronised, but it works sequentially at the level of the neuron: in every step at most one rule is used in each of the neurons. As inspired by the biological findings, the cell sending out spikes may be “closed” for a specific time-period corresponding to the refractory period of a neuron; during this refraction period, the neuron is closed for input and cannot fire token again. Depending on the exact formalisation of the model, the notion of a successful computation is defined together with its output. SN P systems are proved as computationally complete [10]. In addition, many different extensions and modifications of that basic model have been proposed and studied, such as consideration of inhibitory astrocytes and decaying spikes [1, 3].

Different variants of P systems are translated into Petri nets to complement the functional characterisation of their behaviour in [6, 7]. To depict and simulate the behaviour of an SN P system, we introduced the translation of some class of SN P systems into Petri nets. Therefore 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.

Petri nets are bipartite directed graphs consisting of places and transitions. Places represent the objects which indicate the availability of resources, represented by tokens. Thus places can be used to represent neurons and spikes can be indicated with tokens. Transitions are actions which can occur depending upon the availability of resources and thus can be used to 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 modeling paradigm, which allows concurrent systems to be described in a formal yet graphical and well readable way. Furthermore, Petri nets allow for computer aided simulation and what is more, formal analysis of models based on them is also possible.

The relation between SN P systems and Petri nets is emphasized by focusing on modeling simplex stop-and-wait protocol. The SN P system for the protocol problem is constructed and also translated it into equivalent Petri net using the proposed algorithm. Petri nets are widely used for formal specification, analysis and verification of protocols. It is then observed a direct correspondence between the Petri net representation of the proposed model and standard solution based on Petri nets already present in the literature.
The paper is organised as follows, the next two sections give the definitions of SN P system and Petri net. Section 4 gives an algorithm for translating SN P system into equivalent Petri net model. In section 5, we present an SN P system for modeling simplex stop-and-wait protocol and is translated into Petri net model, which is essentially the same as the standard Petri net solution illustrated in [8].

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.
Let the alphabet Σ be the set $\{a_1, a_2, ..., a_n\}$. The letter distribution, φ (w), of a Σ-word w is the n-tuple $\{N_1, \cdots, N_n\}$ with N_i the number of occurrences of a_i in w. The Parikh set, φ (L), of an Σ-language L is $\{\varphi(w) \mid w \text{ in } L\}$. A language L in Σ^* 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 Σ are regular expressions. (ii) If E1, E2 are regular expressions over Σ, then E_1+E_2, E_1E_2 and E_1^* are regular expressions over Σ, and (iii) nothing else is a regular expression over Σ. With each regular expression E, we associate a language L (E).
When $\Sigma = \{a\}$ is a singleton, then the regular expression a^* denotes the set of all strings formed using a .i.e the set $\{\lambda, a, a^2, a^3 ...\}$. The positive closure of a, $a^+ = a^* - \{\lambda\}$. If Σ is a singleton then Parikh set of the language denoted by regular expression E over Σ, L(E) is $\{|w| \mid w \in L(E)\}$.

2. SPIKING NEURAL P SYSTEM

Mathematically, we represent a spiking neural P system (SN P system), of degree $m \geq 1$, in the form $\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma m, syn, i_0)$, where

1. $O = \{a\}$ is the singleton alphabet (a is called spike);
2. $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where
   a) $n_i \geq 0$ is the initial number of spikes contained by the cell;
   b) Ri is a finite set of rules ij associated with neuron σ_i of the following two forms:
   (1)$E / a^r \rightarrow a; t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$; 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
   (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s / L(E)$ for any rule $E / a^r \rightarrow a$; t of type (1) from Ri.

Number of spikes consumed by the rule ij is denoted as con(ij) and number of spikes produced is called pro(ij). In standard SN P systems pro(ij) is either 0 or 1.

3. $syn \subseteq \{1, 2, 3, \ldots, m\} \times \{1, 2, 3, \ldots, m\}$ with (i, i) not in 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 $E/a^r \rightarrow a;t$ are spiking rules, and they are possible only if the neuron contains n spikes such that a^n in L(E) and $n \geq r$. If E $=\varphi$ then the rule is applied only if the neuron contains exactly r spikes. When neuronσ_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 $a^s \rightarrow \lambda$ 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, $a^s \rightarrow \lambda$ is applied only if the neuron contains exactly s spikes.
0-delay SN P system is one where the delay in all the rules of the neurons is zero. Because in this paper we always deal with 0-delay systems, the delay (d = 0) is never specified in the rules.

Definition 2.1 (Configuration) The configuration of the 0-delay SN P system is described by the numbers $C = \{n_1, n_2, \ldots, n_m\}$ representing the number of spikes present in each neuron. The initial configuration C_0 of the system is described by the initial number of spikes present in each 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 2.2 (Vector rule) We define a vector rule v as a mapping with domain Π such that each v(i) is at most one instance of rule from R(i) i.e | v(i) | =0 or 1 where $1 \le i \le m$.

A vector rule v is enabled at a configuration C if, for each neuron σi of Π,

v(i) is either of the form ij: $E/a^r \rightarrow a$ if $n_i \in$ Parikh set of L(E) and $n_i \ge r$ or v(i) is of the form ij: $a^s \rightarrow \lambda$ if n_i is exactly s. If no rule in Ri is enabled then v(i) is i0. If a vector rule v is enabled at a configuration C={n_1, n_2, . . . , n_m} then C can evolve to C'={n'_1, n'_2, . . . , n'_m} such that for every σ_i in Π: $n'_I = n_i - \text{con}(v(i)) + \sum_{(j,i)\in syn} \text{pro}(v(j))$.

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 ⇒ C', if there is a direct transition from C to C' in Π.

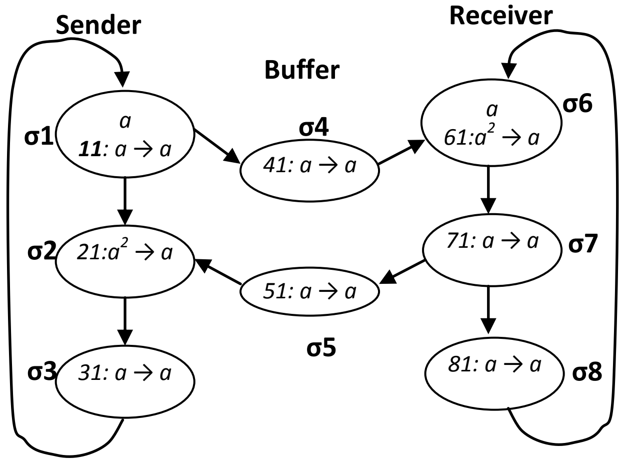

Figure.1: SN P system for simplex stop-and-wait protocol

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. Note that the transition of C is non-deterministic in the sense that there may be different vector rules applicable to C, as described above.

A computation halts if it reaches a configuration where no rule can be used. 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. Because of the non-determinism in using the rules, a given system computes in this way a set of numbers.

3. PETRI NET

A Petri net is a bipartite graph with two types of nodes, place nodes represented with circles containing tokens and transition nodes represented with bars or boxes. The directed arcs connecting places to transitions and transitions to places may be labeled with an integer weight, but if unlabelled are assumed to have a weight equal to 1.

A transition has a certain number of input and output places representing the preconditions and post conditions of the event respectively. A transition is enabled if all of its input places have tokens equal to or greater than the weight of the arc connecting that place to the transition. A transition without any output place is called a sink transition. Note that the firing of a sink transition consumes tokens but does not produce any. Similarly a place without any output transition is called output place.

Many extensions to the simple Petri net model have been developed for various modeling and simulation purposes. These high level Petri nets include coloured Petri nets [5], which allow tokens to have internal structure, and transitions can have a guard function to further constrain their enabling.

Timed Petri nets, in which places and/or transitions may be assigned deterministic/probabilistic time delays [2].

Definition 3.1(Petri net) A Petri net is represented by N= (P, T, A, W, G), where

P = {P_1, P_2, P_3, . . . , P_m} is a finite, nonempty set of places.

T = {T_1, T_2, T_3, . . . , T_n}is a finite, nonempty set of transitions.

A in (P × T) U (T ×P) is a set of directed arcs which connect places with transitions and transitions with places.

W: A→N assigns weight W (f) to elements of $f \in$ A denoting the multiplicity of unary arcs between the connecting nodes.

G: T → {*true, false*}, the guard function maps each transition Ti to Boolean expression, which specifies an additional constraint which must be fulfilled before the transition is enabled.

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. M_0 is the initial marking, the initial number of tokens in each place P_i.

The state or marking of Petri net is changed by the occurrence of transition. Transition T_j is enabled iff its every input place has at least as many tokens as the weight of the input arcs and satisfies the guard function. 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. The pre- and post- of a transition $t \in T$ for all $p \in P$ is defined as:
$PRE_N(t)(p) = W(p, t)$ and $POST_N(t)(p) = W(t, p)$

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. 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. The pre- and post-set of places of transition set U is defined as:
$PRE_N(U) = \sum_{t \in U} PRE_N(t)$ and
$POST_N(U) = \sum_{t \in U} POST_N(t)$
A step U is enabled in a marking M if $M \geq PRE_N(U)$. We denote this by M [U>. Thus, in order for U to be enabled at M, for each place p, the number of tokens in p under M should at least be equal to the total number of tokens that are needed as an input to U, respecting the weights of the input arcs. Moreover, U is a maximal step at M if M [U > and there is no transition t such that M [U + {t}>.
If U is enabled at M, then it can be executed leading to the marking $M0 = M - PRE_N(U) + POST_N(U)$
This means that the execution of U consumes from each place p exactly W (p, t) tokens for each occurrence of a transition $t \in U$ that has p as an input place, and produces in each place p exactly W (t, p) tokens for each occurrence of a transition $t \in U$ with p as an output place.
If the execution of U leads from M to M', we write M [U> M'. Whenever U is a maximal step at M, we will also write M $[U>_{max}$ M'.
A finite sequence $\sigma = U_1 \cdot \cdot \cdot U_n$ of non-empty steps is a step sequence from the initial marking M_0 if there are markings $M_1 \cdot \cdot \cdot M_n$ of N satisfying $M_{i-1}[U_i>_{max}M_i$ for every $i \leq n$. Such an σ is also called a step sequence from M_0 to M_n, and M_n itself is called a reachable marking.
In the same way, we can define step sequences consisting of maximal steps, and markings reachable through such step sequences. Together, they define the maximal concurrency semantics of the Petri net *N*. In this paper we are considering only maximal concurrency semantics of the Petri nets.
A computation of a Petri net *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. 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.

4. SN P SYSTEMS AND PETRI NETS

In this section, we translate a 0-delay SN P system into a behaviourally equivalent Petri net. We also translate some class of Petri nets into equivalent SN P systems.

4.1 SN P System to Petri net

Let $\Pi = (O, \sigma_1, \sigma_2, \sigma_3, . . . , \sigma_m, syn, i_0)$ be a spiking neural P system. Update Π by adding $(i_0, 0)$ to *syn*, to represent the connection between the output neuron and environment. Construct corresponding
Petri net $N_\Pi = (P, T, A, W, G)$ with initial marking M_0, where the various components are defined thus:
1. Each neuron σ_i in an SN P system is represented with a place P_i. Output place P_0 in Petri net corresponds to environment in an SN P system. So add the set of neuron places $\{P_0, P_1, P_2, \cdot\cdot\cdot, P_m\}$ to P. Set $M_0(P_i)=n_i$ and $M_0(P_0) = 0$ where n_i is the initial number of spikes in σ_i.
2. The arcs between a place to transition and transition to place represent an axon. The neuron σ_i spikes using rules *ij*. For each neuron σ_i and each rule *ij* do, if
a) $ij : a^s \rightarrow \lambda$, is the forgetting rule of an SN P system. In Petri net, add a sink transition T_{ij} to T and arc (P_i, T_{ij}) to A with $W(P_i, T_{ij}) = s$. Set $G(T_{ij})$ as true if $M(P_i) = s$.
b) $ij : E / a^r \rightarrow a$. Create and add a transition T_{ij} toT and (P_i, T_{ij}) to A. Set $W(P_i, T_{ij}) = r$. For each $(i, k) \in syn$ add an arc (T_{ij}, P_k) to A with $W(T_{ij}, P_k) = 1$ and set $G(T_{ij})$=*true* if $M(P_i)$ is a member of Parikh set of L(E).
The initial marking of the Petri net N_Π corresponds to initial configuration of an SN P system C_0. To establish the behavioural equivalence of Π and N_Π, we first capture the very tight correspondence between configurations and markings and between enabling of vector rule and steps.

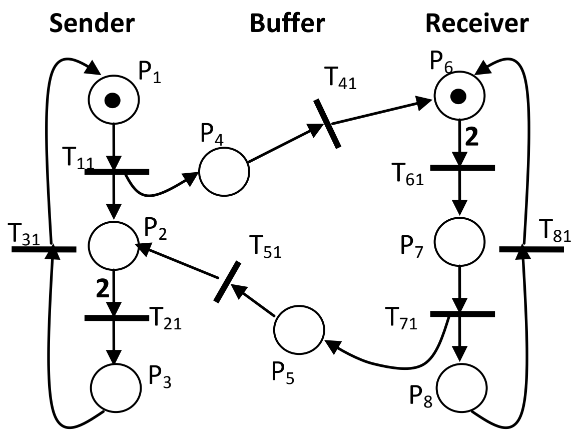

Figure.2: Petri net model for SN P system in Figure.1

Definition 4.1 For each marking M of N_Π the configuration CM is such that for every neuron σ_i for Π, we have $M(P_i)=n_i$.

For each step of transitions U of N_Π, there is vector rule v_U such that for every neuron $\sigma_i \in \Pi$ and rule $v(i) \in v_U$, if t in U is a transition of the form

1. T_{ij} with W $(P_i, T_{ij})=s$ with no out going arcs then $\exists$ a rule $v(i)$ of the form $ij : a^s \rightarrow \lambda$.

2. T_{ij} with input arc from P_i with outgoing arcs then $\exists$ a spiking rule $v(i)$ of the form $ij : E/a^r \rightarrow a$.

If U has no transition beginning with subscript I then $v(i) = i0$.

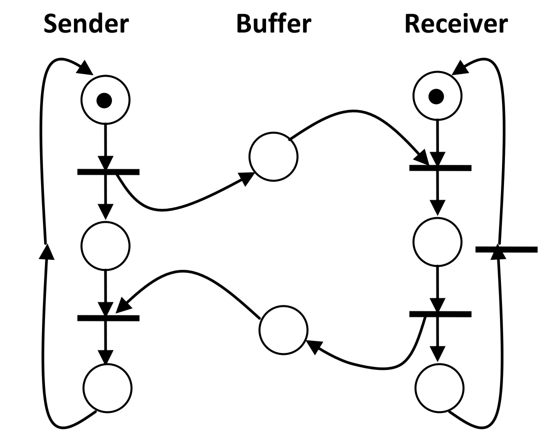

Figure.3: Petri net model for stop-and-wait protocol

Theorem 1.1 Let M is a reachable marking of NΠ, for any execution

1. If M [U>M' then $C_M \Rightarrow C_{M'}$

2. If $C_M \Rightarrow C'$ then there is a step U such that $v = v_U$, M [U>M' and $C_{M'} \Rightarrow C'$.

Proof: Let C_M be a configuration of Π.

(1) We first show that v_U is enabled at CM. As U is enabled at M, by the definition 4.1, for every $t \in U$, $\exists$ a rule $v(i) \in v_U$ such that $v(i)$ is enabled at $C_{M'}$. Hence there is C such that $C_M \Rightarrow C$. Moreover $C = C_{M'}$ follows from the algorithm and definition 4.1.

(2) Let σ_i be a neuron such that $v(i)$ is a rule of any one of the four forms (a) ij: E / $a^r \rightarrow a$; t then the number of spikes in the neuron σ_i, $n_i \geq r$ and $n_i \in L(E)$. That is the rule is enabled. (b) ij: $a^s \rightarrow \lambda$ then by the definition 4.1, $\exists$ a sink transition T_{ij} in U_i with W $(P_i, T_{ij})=s$. (c) *is* : Neuron σ_i spikes and sends a spike to its neighbouring neurons. (d) *i0*: No rule can be used. By the definition 4.1, $\exists$ a transition $U_i \in U$ for $v(i)$. It therefore follows that U is the set of transitions of type U_i where $1 \leq i \leq n$, if $v(i) = i0$ then $U_i=\lambda$. Hence there is M' such that M [U>M'. Moreover $C'=C_{M'}$ follows from the algorithm and definition 4.1.

4.2 Petri net to SN P system

Here we prove that every non-timed basic Petri net N = (P, T, A, W, P_0) with $|I(P_0)| = 1$ (i.e. number of input transitions for place P_0=1) and every transition t $\in$T such that $|I(t)| = 1$ and $W(t, P_j) = 1$. N should also has the property that for every $t_i, t_j \in T$ if $I(t_i) = I(t_j)$ then $O(t_i) = O(t_j)$. N is converted into behaviourally equivalent Spiking Neural P system $\Pi=(O, \sigma_1, \sigma_2, \sigma_3, . . . , \sigma_m, syn, i_0)$ having no timed rules using the following procedure.

1. Set O= $\{a\}$. We have environment in Π for place P_0. Let P_k is the only place connected to P_0. Set i_0 as k

2. For each place $P_i \in P$ except P_0, add $\sigma i=(n_i, R_i)$ with $n_i=M_0(P_i)$ to Π. For each $t \in O(P_i)$ do,

(i). if $O(t) = \varphi$ then add $a^r \rightarrow \lambda$ to R_i else add $E/a^r \rightarrow a$; 0 to R_i where $E = a*$ and $r = W(P_i, t)$

(ii). for each $P_j \in O(t)$ add (i, j) to *syn* if (i, j) is not in *syn*.

We can prove the behavioural equivalence of both systems in a similar way as we proved in Theorem 4.1.

5. SN P SYSTEM FOR SIMPLEX STOP-AND-WAIT PROTOCOL

We consider the problem of modeling simplex stop-and-wait protocol system. The system consists of a sender, receiver and communication channel (buffer). The stop and wait protocol was very easy to implement and runs very quickly and efficiently. It solves the problem of congestion, as only one frame is outstanding at any time, frames can not be lost due to congestion and the receiver will not be swamped by the sender. It assumes an error free communication channel. It is easy to see that if a frame or an acknowledgment gets lost or damaged, a deadlock situation will occur where neither the sender nor the receiver can advance; they will be thrown into infinite loops. Specifically, the sender has three states: "ready to send", "wait for ack (acknowledgment) and "ack received". The receiver also has three states, "ready to receive", "packet received" and "ack sent". The buffer has two states: “has packet” and “has ack”. In state “ready to send”, the sender executes the operation “send packet” and moves to state “wait for ack”; in state “wait for ack”, if the buffer “has ack”, the sender executes the operation “receive ack”, and moves to “ack received”, from this state it moves back to state “ready to send”. Similarly, the receiver is in state “ready to receive” and if the buffer “has packet”, the receiver executes the operation “receive”, and moves to state “packet received” and then executes the operation “send ack” and moves to state “sent ack”. From this state the receiver goes back to “ready to receive”.

In this protocol, there can only be one outstanding frame at a time so no sequence numbers are required and the acknowledgment the receiver sends back to neuron 2, which has two spikes, so it fires using the rule $a^2 \to a$. The rule $a^2 \to a$ is fired only if the sender is waiting for the ack and buffer has an ack and the receiver again moves to the state 8. The rules $a^2 \to a$ in neuron 2 and 6 provide synchronisation between sender and receiver.

Now let us consider the Petri net model of the simplex stop-and-wait protocol system given in [8] and it is reported in Figure.3. If we construct equivalent Petri net for the SN P system of Figure.1 using the procedure in section 4, we get a Petri net of Figure.2 which is equivalent to Petri net of Figure.3. In other words, we observe a “direct” correspondence between the SN P systems representations and the sender is nothing more than an empty frame, as there is no other possibility than acknowledging the only frame sent. Another frame will not be sent until this acknowledgment is received.

In order to model the simplex stop-and-wait protocol, we consider an SN P system in Figure.1 with 8 neurons labeled in a one-to-one manner with values in σ_1 to σ_8. The neurons 1, 2 and 3 respectively represent “ready to send”, “wait for ack (acknowledgment)” and “ack received” states of the sender.

Similarly neurons 4 and 5 represent “has packet” and “has ack” states of the buffer respectively. The neurons 6, 7 and 8 denote respectively the “ready to receive”, “packet received” and “ack sent” states of the receiver. The operations are represented by rules which are labeled with unique number I or each neuron σ_i. Initially neurons 1 and 6 have one spike each representing that sender is initially in “ready to send” state and receiver is in “ready to receive” state. The neurons 1 fire in the first step using the rule 11 : $a \to a$ representing the operation ”send packet” and the sender moves to the state “wait for ack” by sending a spike to neurons 2 and 4. In the next step the neuron 4 fires and sends its spike to neuron 6. As neuron 6 has two spikes presenting that it is ready to receive and packet is in the buffer, it fires using its rule 61: $a^2 \to a$ which represents the “receive packet” operation. The neuron 6 send spikes to neuron 7 which fires using a rule $a \to a$ representing the operation “send ack” and sends spikes to neuron 5 and 8. In the next steps both the neurons fire. The neuron 5 sends its spike Petri nets representation.

Conclusion

In this paper we have proposed an approach to the modeling of the behaviour of membrane systems through a class of Petri nets. We gave first a formal translation for a basic class of spiking neural P systems, and argued that the structure of the concurrent computations of such SN P systems is faithfully reflected by the maximal concurrency semantics of the corresponding Petri nets. Clearly, one could simply use the basic model of Petri nets and simulate to study the behaviour of spiking neural P systems.

We have also given translation for restricted class of Petri nets into SN P systems. We have constructed an SN P system for simplex stop-and-wait protocol and translated it into equivalent Petri net using the proposed algorithm and observed that it is equivalent to the standard Petri net representation described in [8]. It would be interesting to consider different variations and normal forms of SN P systems and find out the suitable class of Petri nets which could simulate these variations and normal forms.

References

[1] Binder A, Freund R, Oswald M, Vock L, “Extended Spiking Neural P systems with Excitatory and Inhibitory Astrocytes”, Proceedings of the 8th WSEAS international conference on Evolutionary Computing, British Columbia, Canada , June 19-21, 2007.

[2] Fred, Bowden D J, “A Brief Survey and Synthesis of the Roles of Time in Petri nets”, Mathematical and Computer Modelling, vol.31, No.10-12, pp.55-68, 2000.

[3] Freund R, Ionescu M, Oswald M, “Extended Spiking Neural P Systems with Decaying Spikes and-or Total Spiking”, ACME FCT Workshop, Budapest, 2007.

[4] Ionescu M, Paun Gh, Yokomori T, “Spiking Neural P Systems”, Fundamenta Informaticae, vol.71, No.2-3, pp.279-308, 2006.

[5] Jenson K, “Coloured Petri nets: Basic Concepts, Analysis, Methods and Practical Use”, EACTS, Monographs on Theoretical Computer Science. Springer –Verlag, 1992.

[6] Kleijn J, Koutny M, Rozenberg G “Process Semantics for Membrane System”, Journal of Automata, Languages and Combinatorics , vol 11, pp.321-340, 2006.

[7] Kleijn J, Koutny M “A Petri net model for membrane system with dynamic structure”, Journal of Natural Computing, 2008.

[8] Munina Yusufu, “Petri nets”, http://www.cas.mcmaster. /sartipi/course/cas707/w07/slides/Mar13-PetriNets-Munina.pdf

[9] Paun Gh, “Computing with Membranes”, Journal of Computer and System Sciences, Vol.61, pp.108-143, 2000.

[10] Paun A, Paun Gh, “Small Universal Spiking Neural P Systems”, Journal of Biosystems, Elsevier, Vol.90, pp.48-60, 2007.