

# Automated Verification of Stochastic Spiking Neural P Systems

Bogdan Aman<sup>(✉)</sup> and Gabriel Ciobanu

Institute of Computer Science, Romanian Academy, Iași, Romania  
bogdan.aman@gmail.com, gabriel@info.uaic.ro

**Abstract.** In this paper we consider stochastic spiking neural P systems, a class of distributed parallel neural-like computing models. We translate a restricted variant of the stochastic spiking neural P systems using uniform distribution into a network of timed automata, proving that such a translation preserves faithfully their behaviours. This relationship allows the verification of several kinds of properties (both qualitative and quantitative) using the statistical model checking extension of the complex software tool UPPAAL.

## 1 Introduction

Membrane computing [13] is a known branch of natural computing that aims to abstract computing ideas and formal models from the structure and functioning of living cells, as well as from the organization of cells in tissues, organs (brain included) or other higher order structures such as colonies of cells (e.g., of bacteria) [1]. A structure is represented by a set of regions, each delimited by a surrounding membrane, and arranged in a tree or a graph form. Multisets of objects are distributed inside these regions, and they can be modified or moved between adjacent/connected compartments. Objects represent the formal counterpart of the molecular species (spikes, ions, proteins, etc.) floating inside cellular compartments, and are described by means of strings over a given alphabet. Evolution rules represent the formal counterpart of chemical reactions, and are given in the form of rewriting rules that operate on objects. The models considered, called membrane systems (P systems), are parallel, distributed computing models, processing multisets of symbols in cell-like compartmental architectures. These models have been applied to the description of biological systems [10, 11].

Spiking neural (SN) P systems represent a class of distributed parallel computing models inspired from the way neurons communicate with each other by means of electrical impulses (see Fig. 1), where there is a synapse between each pair of connected neurons. Roughly, a spiking neural P system consists of a set of neurons placed in the nodes of a directed graph, where neurons send signals (spikes, denoted by the symbol  $a$ ) along synapses (arcs of the graph). Stochastic spiking neural P systems are obtained from spiking neural P systems by associating to each spiking rule a firing time that indicates how long an enabled rule

waits before it is executed. Such firing times are random variables (abstractions for the concept of chance) whose probability distribution functions have domain contained in  $\mathbb{R}^+$ .

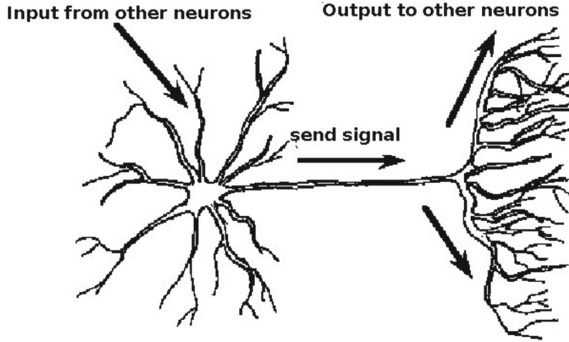


Fig. 1. Communication between neurons

The presence of unreliable components in spiking neural P system can be considered in many different aspects (e.g., in the form of a stochastic delays of the spiking rules [8], or the stochastic loss of spikes [15]). The presence of unreliable components pose an important constrains on the possible modelling and verification of spiking neural P system. In this paper we provide a formally correct algorithm for translating systems described in stochastic spiking neural P systems of [8] into a class of timed safety automata. This connection allows the verification of several kinds of properties, both qualitative and quantitative, using the statistical model checking extension of the UPPAAL software tool.

## 2 Stochastic Spiking Neural P Systems

Some notations and basic definitions are shortly presented.

The set of non-negative integers is denoted by  $\mathbb{N}$ . Given a finite alphabet  $V = \{a_1, \dots, a_n\}$ , the free monoid generated by  $V$  under the operation of concatenation is denoted by  $V^*$ . The elements of  $V^*$  are called strings, and the empty string is denoted by  $\lambda$ . The set of all non-empty strings over  $V$  is denoted by  $V^+$ . When  $V = \{a\}$  is a singleton, then we write simply  $a^*$  and  $a^+$  instead of  $\{a\}^*$  and  $\{a\}^+$ , respectively.

A regular expression  $E$  over an alphabet  $V$  is defined as follows:

$$E = \lambda \mid a \mid (E)(E) \mid (E) \cup (E) \mid (E)^+, \text{ where } a \in V.$$

$E^* = (E)^+ \cup \{\lambda\}$ . We associate a language  $L(E)$  to each expression  $E$ :

$$L(E) = \begin{cases} \{\lambda\} & \text{if } E = \lambda; \\ \{a\} & \text{if } E = a; \\ L(E_1)L(E_2) & \text{if } E = (E_1)(E_2); \\ L(E_1) \cup L(E_2) & \text{if } E = (E_1) \cup (E_2); \\ L((E_1)^+) & \text{if } E = (E_1)^+. \end{cases}$$

Some parentheses can be omitted when writing a regular expression. More details can be found in [14].

We use a restricted version of the stochastic spiking neural P system presented in [8], considering only uniform distribution up to a given bound.

**Definition 1.** *A stochastic spiking neural P system of degree  $m \geq 1$  is defined by  $\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, \text{out})$ , where:*

- $O = \{a\}$  is the singleton alphabet ( $a$  is called spike);
- $\sigma_1, \dots, \sigma_m$  are neurons, of the form

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

- (a)  $n_i \geq 0$  is the initial number of spikes contained in  $\sigma_i$ ;
- (b)  $R_i$  is a finite set of rules of the following two forms:
  - (1)  $E/a^c \rightarrow a; F(d)$ , where  $E$  is a regular expression over  $a$ , and  $c \geq 1$ , and  $F$  is a probability distribution function with domain  $[0, d]$ ;
  - (2)  $a^s \rightarrow \lambda; F'(d)$ , for  $s \geq 1$ , with the restriction that for each rule  $E/a^c \rightarrow a; F'$  of type (1) from  $R_i$ , we have  $a^s \notin L(E)$ , and  $F'$  is a probability distribution function with domain  $[0, d]$ ;
- $\text{syn} \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\} \times \mathbb{N}$  with  $i \neq j$  for each  $(i, j, r) \in \text{syn}$ ,  $1 \leq i, j \leq m$  (synapses between neurons);
- $\text{out} \in \{1, 2, \dots, m\}$  indicates the output neuron.

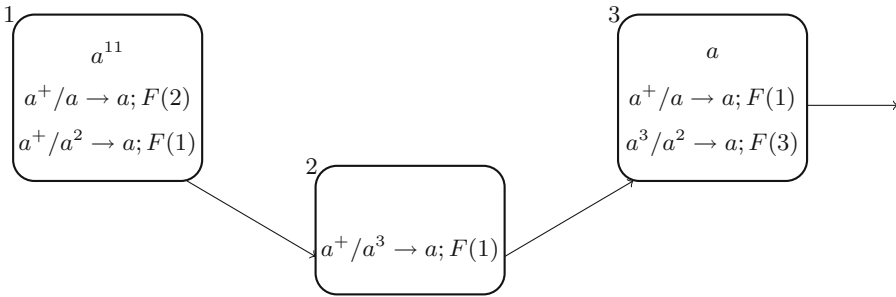
The rules of type (1) are called *spiking rules*, and are applied as follows: if the neuron  $\sigma_i$  contains  $k$  spikes, and  $a^k \in L(E)$ ,  $k \geq c$ , then the rule  $E/a^c \rightarrow a; F'$  can be applied. This means removing  $c$  spikes from neuron  $\sigma_i$ , and producing 1 spike. The rules of type (2) are called *forgetting rules* and are applied as follows: if the neuron  $\sigma_i$  contains exactly  $s$  spikes, then the rule  $a^s \rightarrow \lambda; F''$  from  $R_i$  can be used, meaning that  $s$  spikes are removed from neuron  $\sigma_i$ .

From the moment in which a rule is enabled up to the moment when the rule fires, a random amount of time elapses, whose probability distribution is specified by a function  $F$  associated to the rule (different rules may have associated different distributions). Once the rule fires, the update of the number of spikes in the neuron, the emission of spikes and the update of spikes in the receiving neurons are all simultaneous and instantaneous events. Multiple rules may be simultaneously enabled in the same neuron. Whenever multiple enabled rules in a neuron have the same random firing time, the order of firing is randomly chosen, with a uniform probability distribution across the set of possible firing orders.

The initial configuration of the system is  $C_0 = \{n_1, \dots, n_m\}$ , where  $n_1, \dots, n_m$  are the numbers of spikes present in each neuron. During the computation, a configuration  $C = \{n'_1, \dots, n'_m\}$  is described by the number of spikes  $n'_i$  present in each neuron  $\sigma_i$ , for  $1 \leq i \leq m$ . Using the rules described above, we can define transitions among the configurations of a system. Notice that, because of the way the firing of the rules has been defined, in general there is no upper bound on how many rules fire for each transition. For two configurations

$C_1, C_2$  of  $\Pi$  we denote by  $C_1 \xrightarrow{r_j} C_2$  the effect of applying a rule  $r_j$  of a neuron. Also  $C_1 \Rightarrow C_2$  denotes the fact that there is a direct transition from  $C_1$  to  $C_2$  in  $\Pi$  in which at most one rule was applied in each neuron, followed by moving to the next time step. The reflexive and transitive closure of the relation  $\Rightarrow$  is denoted by  $\Rightarrow^*$ . Any sequence of transitions starting in the initial configuration  $C_0$  is called a computation. A computation halts if it reaches a configuration  $C_i$  where no rule can be used.

*Example 1.* In what follows we present a graphical form of stochastic spiking neural P systems. Here we just introduce the example without emphasizing on its behaviour (we will consider latter this aspect). In Fig. 2, each neuron is represented by an oval marked with a label and having inside both its current number of spikes and its rules. The synapses linking the neurons are represented by directed arrows, while a short directed arrow pointing to (from) the environment identifies the output (input) neuron. In the following example we consider only an output neuron (and so the input synapse is not drawn).



**Fig. 2.** A simple example of a stochastic SN P system

The system consists of three neurons labelled by 1, 2, 3 in which neuron 3 is the output one. In an initial configuration  $C_0$ , neurons 1 and 3 are ready to fire. The spike of neuron 3 leaves it empty, and unable to spike again before receiving a new spike. Neuron 2 cannot fire until it succeeds to collect exactly 3 spikes. The computation continues until consuming all spikes from all neurons.

### 3 Networks of Timed Automata

Timed automata [2] extended with integer variables, structured data types, user defined functions, broadcast, urgent channels and channel synchronization have been used by several software tools for simulation and verification of various systems with time.

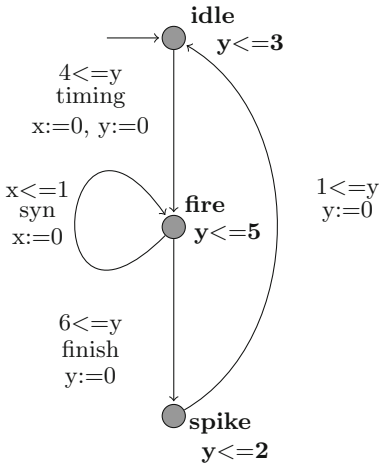
*Syntax.* We assume a finite set of real-valued variables  $\mathcal{C}$  ranged over by  $x, y$  standing for clocks, a set of clock resets ranged by  $r, r_i$ , and a finite alphabet  $\Sigma$  ranged over by  $a, b$  standing for actions. A clock constraint  $g$  is a conjunctive formula of constraints of the form  $x \sim m$  or  $x - y \sim m$ , for  $x, y \in \mathcal{C}$ ,  $\sim \in \{\leq, <, =, >, \geq\}$ , and  $m \in \mathbb{N}$ . The set of clock constraints is denoted by  $\mathcal{B}(\mathcal{C})$ .

**Definition 2.** A **timed safety automaton**  $\mathcal{A}$  is a tuple  $\langle N, n_0, E, I \rangle$ , where

- $N$  is a finite set of nodes;
- $n_0$  is the initial node;
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$  is the set of edges;
- $I : N \rightarrow \mathcal{B}(\mathcal{C})$  assigns invariants to nodes.

$n \xrightarrow{g,a,r} n'$  is a shorthand notation for  $\langle n, g, a, r, n' \rangle \in E$ . Node invariants are restricted to constraints of the form  $x \leq m$  or  $x < m$ , where  $m \in \mathbb{N}$ .

A simple example of a timed safety automaton is depicted in Fig. 3.



**Fig. 3.** Timed safety automata

A timed safety automata is a graph having a finite set of nodes and a finite set of labelled transitions, using real time clocks. The clocks are initialized with zero when the system starts, and then increased synchronously with the same rate. The behaviour of the automaton is restricted by using clock constraints, i.e. guards on transitions, and *node invariants* (e.g., see Fig. 3). An automaton is allowed to stay in a node as long as the timing conditions of that node are satisfied. A transition can be taken when the transition guards are satisfied. When a transition is taken, clocks may be reset to zero.

$\mathcal{A}_1, \dots, \mathcal{A}_n$  combined into a single system using the CCS-like parallel composition operator and with all internal actions hidden. Synchronous communication inside the network is by handshake synchronization of input and output actions. In this case, the action alphabet  $\Sigma$  consists of  $a?$  symbols (for input actions),  $a!$  symbols (for output actions), and  $\tau$  symbols (for internal actions). A detailed example is found in [12].

*Networks of Timed Automata.* A network of timed automata is the parallel composition  $\mathcal{A}_1 \mid \dots \mid \mathcal{A}_n$  of a set of timed automata

A network can perform both delay and action transitions. An action transition is enabled if the clocks and variables assignment satisfies all guards on the corresponding edges. In synchronization transitions, the resets on the edge with an output label are performed before the resets on the edge with an input label. To model urgent synchronization transitions that have priority with respect to the delay transitions, a notion of urgent channels is used. On urgent channels it is not possible to delay an execution whenever such an execution is possible. One-to-many synchronizations are possible using broadcast channels: an edge with synchronization label  $a!$  emits a broadcast and any enabled edge with synchronization label  $a?$  synchronizes with the emitting automata.

Let  $u, v, \dots$  denote clock assignments mapping  $\mathcal{C}$  to  $\mathbb{R}_+$  of non-negative real numbers.  $g \models u$  means that the clock values  $u$  satisfy the guard  $g$ . For  $d \in \mathbb{R}_+$ , the clock assignment mapping all  $x \in \mathcal{C}$  to  $u(x) + d$  is denoted by  $u + d$ . Also, for  $r \subseteq \mathcal{C}$ , the clock assignment mapping all clocks of  $r$  to 0 and agreeing with  $u$  for the other clocks in  $\mathcal{C} \setminus r$  is denoted by  $[r \mapsto 0]u$ . Let  $n_i$  stand for the  $i$ th element of a node vector  $n$ , and  $n[n'_i/n_i]$  for the vector  $n$  with  $n_i$  being substituted with  $n'_i$ .

A network state is a pair  $\langle n, u \rangle$ , where  $n$  denotes a vector of current nodes of the network (one for each automaton), and  $u$  is a clock assignment storing the current values of all network clocks and integer variables.

**Definition 3.** *The operational semantics of a timed automaton is a transition system where states are pairs  $\langle n, u \rangle$  and transitions are defined by the rules:*

- $\langle n, u \rangle \xrightarrow{d} \langle n, u + d \rangle$  if  $u \in I(n)$  and  $(u + d) \in I(n)$ , where  $I(n) = \bigwedge I(n_i)$ ;
- $\langle n, u \rangle \xrightarrow{\tau} \langle n[n'_i/n_i], u' \rangle$  if  $n_i \xrightarrow{g, \tau, r} n'_i$ ,  $g \models u$ ,  $u' = [r \mapsto 0]u$  and  $u' \in I(n[n'_i/n_i])$ ;
- $\langle n, u \rangle \xrightarrow{\tau} \langle n[n'_i/n_i][n'_j/n_j], u' \rangle$  if there exist  $i \neq j$  such that
  1.  $n_i \xrightarrow{g_i, a^?, r_i} n'_i$ ,  $n_j \xrightarrow{g_j, a^!, r_j} n'_j$ ,  $g_i \wedge g_j \models u$ ,
  2.  $u' = [r_i \mapsto 0][r_j \mapsto 0]u$  and  $u' \in I(n[n'_i/n_i][n'_j/n_j])$ .

## 4 Relating Stochastic SN P Systems to Timed Automata

In this section we present an algorithmic translation of stochastic spiking neural P systems into timed safety automata, and prove that such a timed safety automata has a bisimilar behaviour with the initial stochastic spiking neural P system. This allows the use of existing tools such as UPPAAL for the verification of complex systems of neurons.

*Building a Timed Safety Automaton for each Neuron:* Given a neuron  $\sigma_i = (n_i, R_i)$  of a stochastic spiking neural P system  $\Pi$ , we associate to it several timed safety automata.

- For each rule  $r_{ij} : E/a^c \rightarrow a; F(d) \in R_i$  we associate an automaton  $\mathcal{A}_{ij} = \langle N_i, n_{ij}, E_{ij}, I_{ij} \rangle$ , where the components are as follows:
  - $N_i = \{n_{ij}, n_{ij}\text{-fired}\}$   
The node  $n_{ij}$  denotes that in neuron  $i$  exists a rule  $r_{ij}$ , while the node  $n_{ij}\text{-fired}$  illustrates that the neuron  $i$  fired the rule  $r_{ij}$ .
  - $I(n_{ij}) = \{x \leq d\}$ ,  $I(n_{ij}\text{-fired}) = \{x \leq 0\}$   
The nodes  $n_{ij}$  and  $n_{ij}\text{-fired}$  should be exited before a maximum of  $d$  and 0, respectively, units of time have elapsed.
  - $E_{ij} = \{n_{ij}, E, r[i][j]?, \{n_i = n_i - c, x = 0\}, n_{ijc}\text{-fired}\}$   
The transition  $\{n_{ij}, E, r[i][j]?, \{n_i = n_i - c, x = 0\}, n_{ijc}\}$  illustrates the fact that when a rule  $r_{ij}$  is executed in neuron  $i$  (denoted by the synchronization on urgent channel  $r[i][j]?$  and the fulfilment of expression  $E$ ), then  $c$  spikes are removed from  $n_{ij}$  and the local clock  $x$  is reset to 0 in order to model the delay according to the distribution  $F(d)$ . Using urgent

channels illustrates the fact that from all rules of a neuron one will be selected nondeterministically.

To simulate the continuation of the rule we have three cases:

- (1)  $E_{ij} = E_{ij} \cup \{n_{ij\_fired}, syn[i][1]?, n_{ij}\}$   
 The transition  $\{n_{ij\_fired}, syn[i][1]?, n_{ij}\}$  illustrates the fact that the spike created by rule  $r_{ij}$  is sent on all outgoing synapses (illustrated by the broadcast channel  $syn[i][1]$ ). Graphically the obtain automaton can be represented as in Fig. 4.

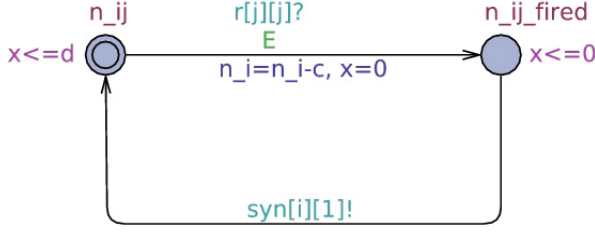


Fig. 4. An automaton associated to a rule  $r_{ij} : E/a^c \rightarrow a; F(d)$

- (2)  $E_{ij} = E_{ij} \cup \{n_{ij\_fired}, n_{ij}\}$   
 This case is similar with the previous case, except that there is no outgoing synapse (illustrated by the missing of the broadcast channel  $syn[i][1]$ ) as illustrated in Fig. 5.

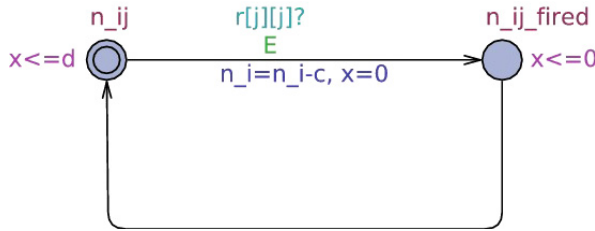
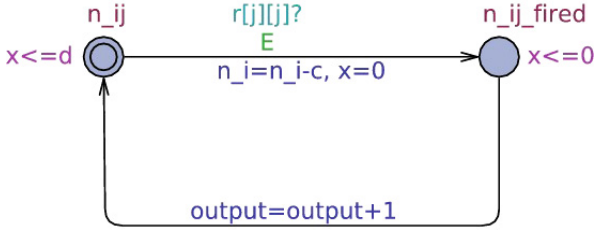


Fig. 5. An automaton associated to a rule  $r_{ij} : E/a^c \rightarrow a; F(d)$

- (3)  $E_{ij} = E_{ij} \cup \{n_{ij\_fired}, output = output + 1, n_{ij}\}$   
 This case is similar with the case (1), except that there is no outgoing synapse (illustrated by the missing of the broadcast channel  $syn[i][1]$ ) but the current neuron is the output neuron (illustrated by the update  $output = output + 1$ ). Graphically this case can be represented as in Fig. 6.



**Fig. 6.** A transition associated to a rule  $r_{ij} : E/a^c \rightarrow a^p; d$

– for each rule  $r_{ij} : a^s \rightarrow \lambda$  we associate an automaton  $\mathcal{A}_{ij} = \langle N_i, n_{ij}, E_{ij}, I_{ij} \rangle$ , where the components are as follows:

- $N_i = \{n_{ij}, n_{ij\_fired}\}$

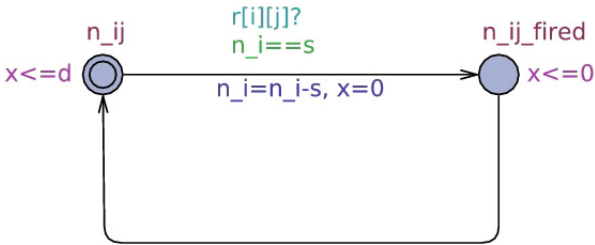
The node  $n_{ij}$  denotes that in neuron  $i$  exists a rule  $r_{ij}$ , while the node  $n_{ij\_fired}$  illustrates that the neuron  $i$  fired the rule  $r_{ij}$ .

- $I(n_{ij}) = \{x \leq d\}$

The node  $n_{ij}$  should be exited before a maximum of  $d$  units of time have elapsed.

- $E_{ij} = \{n_{ij}, n_i == s, r[i][j]?, \{n_i = n_i - s, x = 0\}, n_{ij\_fired}\} \cup \{n_{ij\_fired}, \dots, n_{ij}\}$

The transition  $\{n_{ij}, n_i == s, r[i][j]?, \{n_i = n_i - s, x = 0\}, n_{ij\_fired}\}$  describes that  $s$  spikes are removed from  $n_i$ , if  $n_i$  contains exactly  $s$  spikes, and the local clock  $x$  is reset to 0 whenever a forgetting rule  $r_{ij}$  is executed in neuron  $i$  (denoted by the synchronization on urgent channel  $r[i][j]?$ ). The transition  $\{n_{ij\_fired}, \dots, n_{ij}\}$  illustrates that in the next step the neuron will be able to fire again. Graphically the automaton is represented in Fig. 7.



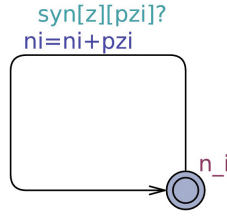
**Fig. 7.** A transition associated to a rule  $r_j : a^s \rightarrow \lambda$

– For each neuron  $n_i$  we associate an automaton  $\mathcal{A}_i = \langle N_i, n_i, E_i, I_i \rangle$ , where  $N_i = \{n_i\}$ ,  $E_i = \emptyset$ ,  $I_i = \emptyset$ . The components  $N_i$ ,  $E_i$  and  $I_i$  are updated depending on the structure of  $\sigma_i$  and the incoming/outgoing synapses:

- for each incoming synapse  $(z, i)$  we have:
  - \*  $E_i = E_i \cup \{n_i, \text{syn}[z][pzi]!, n_i = n_i + pzi, n_i\}$ ;

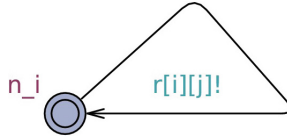
If on synapse  $(z, i)$  are received  $pzi$  spikes on the broadcast channel  $\text{syn}$ , then the number of spikes from neuron  $n_i$  is incremented with  $pzi$ . Graphically this transition can be represented as in Fig. 8.





**Fig. 8.** A transition associated to an incoming synapse  $(z, i, w_{zi})$

- for each rule  $r_{ij} \in R_i$  we have:
  - \*  $E_i = E_i \cup \{n_i, r[i][j]!, n_i\}$ ;
 This transition signifies the fact that a rule  $r_{ij}$  of neuron  $n_i$  will be executed if it synchronizes on the urgent channel  $r[i][j]$ . Graphically this transition can be represented as in Fig. 9.



**Fig. 9.** An automaton associated to a rule  $r_{ij}$

Building a timed automaton for each neuron leads to the next result about the equivalence between a stochastic spiking neural P systems  $\Pi$  with the initial configuration  $C_0$  and its corresponding timed safety automaton  $\mathcal{A}_\Pi$  in the initial state  $\langle n_{C_0}, u_{C_0} \rangle$  (i.e.,  $(\mathcal{A}_\Pi, \langle n_{C_0}, u_{C_0} \rangle)$ ). Their transition systems differ not only in transitions, but also in states. Thus, we adapt the notion of bisimilarity.

**Definition 4.** A symmetric relation  $\sim$  over stochastic spiking neural P systems and the corresponding timed safety automata, is a bisimulation if whenever  $(C, (\mathcal{A}_\Pi, \langle n_C, u_C \rangle)) \in \sim$ :

- if  $C \xrightarrow{r_{ij}^c} C'$ , then  $\langle n_C, u_C \rangle \xrightarrow{\tau} \langle n_{C'}, u_{C'} \rangle$  and  $(C', (\mathcal{A}_\Pi, \langle n_{C'}, u_{C'} \rangle)) \in \sim$  for some  $C'$ .
- if  $C \xrightarrow{r_{ij}^p} C'$ , then  $\langle n_C, u_C \rangle \xrightarrow{\tau} \langle n_{C'}, u_{C'} \rangle$  and  $(C', (\mathcal{A}_\Pi, \langle n_{C'}, u_{C'} \rangle)) \in \sim$  for some  $C'$ .
- if  $C \xrightarrow{d} C'$ , then  $\langle n_C, u_C \rangle \xrightarrow{d} \langle n_{C'}, u_{C'} \rangle$  and  $(C', (\mathcal{A}_\Pi, \langle n_{C'}, u_{C'} \rangle)) \in \sim$  for some  $C'$ , where  $u_{C'} = u_C + d$ .

Having defined bisimulation, we can state our main theorem as follows.

**Theorem 1.** Given a stochastic spiking neural P system  $\Pi$  with initial configuration  $C_0$ , there exists a timed safety automaton  $\mathcal{A}_\Pi$  with a bisimilar behaviour. Formally,  $C_0 \sim (\mathcal{A}_\Pi, \langle n_{C_0}, u_{C_0} \rangle)$ .

*Proof (Sketch).* The construction of the timed safety automaton simulating a given stochastic spiking neural P system is presented above.

A bisimilar behaviour is given by:

- when execution starts, the global clock of the stochastic spiking neural P system and the local clocks of the corresponding timed automata are set to 0;
- the application of a rule in a neuron is matched by two  $\tau$  edges obtained by translation (a  $\tau$  edge corresponds to the consumption/production of spikes);
- the passage of time is similar in both formalisms: in stochastic spiking neural P system the global clock is used to decrement by  $d$  all timers in the configuration when no rule is applicable, while in the timed automata all local clocks are decremented synchronously with the same value  $d$  when no edge can be taken.

Thus, the size of a timed safety automata  $\mathcal{A}_\Pi$  is polynomial with respect to the size of a stochastic spiking neural P system  $\Pi$ , and the state spaces have the same number of states.

*Reachability Analysis.* One of the most useful question to ask about a timed automaton is the reachability of a given set of final states. Such final states may be used to characterize safety properties of a system.

**Definition 5.** We write  $\langle n, u \rangle \rightarrow \langle n', u' \rangle$  whenever  $\langle n, u \rangle \xrightarrow{\sigma} \langle n', u' \rangle$  for  $\sigma \in \Sigma \cup \mathbb{R}_+$ . For an automaton with initial state  $\langle n_0, u_0 \rangle$ ,  $\langle n, u \rangle$  is reachable if and only if  $\langle n_0, u_0 \rangle \rightarrow^* \langle n, u \rangle$ . More generally, given a constraint  $\phi \in \mathcal{B}(\mathcal{C})$  if  $\langle n, u \rangle$  is reachable for some  $u$  satisfying  $\phi$  then a state  $\langle n, \phi \rangle$  is reachable.

Invariant properties can be specified using clock constraints in combination with local properties on nodes. The reachability problem is decidable [7].

The reachability problem can be also defined for stochastic SN P systems.

**Definition 6.** We write  $C \rightarrow C'$  if  $C \xrightarrow{r_j^c} C'$  or  $C \xrightarrow{r_j^p} C'$  or  $C \xrightarrow{d} C'$ . Starting from a configuration  $C_0$ , a configuration  $C_1$  is reachable if and only if  $C_0 \rightarrow^* C_1$ .

The following result is a consequence of Theorem 1.

**Corollary 1.** For a stochastic spiking neural P system, the reachability problem is decidable.

*Bisimulation.* Two timed automata are defined to be timed bisimilar in [7] if and only if they perform the same action transitions and reach bisimilar states.

**Definition 7.** A symmetric relation  $\mathcal{R}$  over the timed automata and the alphabet  $\Sigma \cup \mathbb{R}_+$ , is a bisimulation if:

- for all  $(s_1, s_2) \in \mathcal{R}$ , if  $s_1 \xrightarrow{\sigma} s'_1$  for  $\sigma \in \Sigma \cup \mathbb{R}_+$  and  $s'_1$ , then  $s_2 \xrightarrow{\sigma} s'_2$  and  $(s'_1, s'_2) \in \mathcal{R}$  for some  $s'_2$ .

**Proposition 1.** [9] Timed bisimulation is decidable.

In a similar way we define the bisimulation over configurations of stochastic spiking neural P systems.

**Definition 8.** A symmetric relation  $\mathcal{R}$  over configurations of stochastic spiking neural P systems, is a bisimulation if:

- for all  $(C_1, C_2) \in \mathcal{R}$ , if  $C_1 \xrightarrow{r_j}_c C'_1$  for some  $C'_1$ , then  $C_2 \xrightarrow{r_j}_c C'_2$  and  $(C'_1, C'_2) \in \mathcal{R}$  for some  $C'_2$ .
- for all  $(C_1, C_2) \in \mathcal{R}$ , if  $C_1 \xrightarrow{r_j}_p C'_1$  for some  $C'_1$ , then  $C_2 \xrightarrow{r_j}_p C'_2$  and  $(C'_1, C'_2) \in \mathcal{R}$  for some  $C'_2$ .
- for all  $(C_1, C_2) \in \mathcal{R}$ , if  $C_1 \xrightarrow{t} C'_1$  for  $t \in \mathbb{N}$  and some  $C'_1$ , then  $C_2 \xrightarrow{t} C'_2$  and  $(C'_1, C'_2) \in \mathcal{R}$  for some  $C'_2$ .

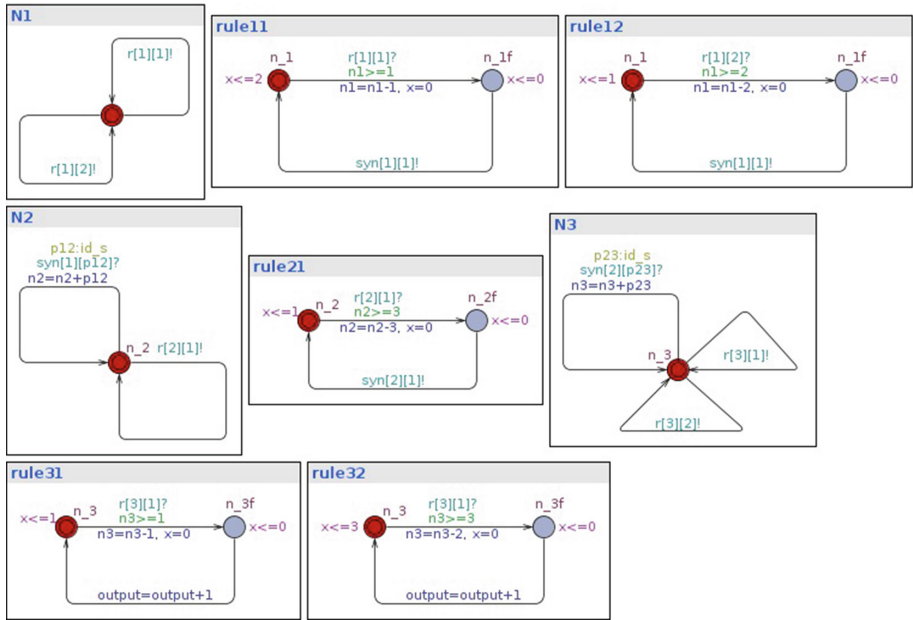
The following result is a consequence of Theorem 1.

**Corollary 2.** *For two configurations of stochastic spiking neural P systems, timed bisimulation is decidable.*

## 5 Verification of Stochastic Spiking Neural P Systems

In this section we present the automated verification of the stochastic spiking neural P system by using the software tool UPPAAL (<http://www.uppaal.org/>). Such a verification is possible due to the translation of stochastic spiking neural P systems into timed safety automata presented in the previous section.

We start from the stochastic spiking neural P system described in Example 1, and translate it into the timed safety automata described in Fig. 10.



**Fig. 10.** A simple example modelled in UPPAAL

UPPAAL allows the automated verification of several properties involving several thousands of possible states, very difficult to be validated by any experimental effort. In this way we show how it is possible to prove/verify certain

complex properties of complex biological systems modelled by stochastic spiking neural P systems. This can be done without the high expenses required by the experimental work in laboratories leading sometimes to wrong conclusions.

The model checking approach uses various techniques to automatically and efficiently check a given system against specified formulas. The formulas can be of two types path formulae (quantify over paths or traces of the model) and state formulae (individual states). Path formulae can be classified into reachability ( $E \langle \rangle \phi$ ), safety ( $A [] \phi$  and  $E [] \phi$ ) and liveness ( $A \langle \rangle \phi$  and  $\phi \rightsquigarrow \psi$ ), where  $\phi$  and  $\psi$  are boolean expressions over predicates on nodes and integer variables.

Reachability properties are used to check whether there exist a path starting at an initial state, such that  $\phi$  is eventually satisfied along that path. Safety properties are used to verify that something bad will never happen, while liveness properties check whether the system always progresses.

We present various properties that could be analyzed and verified for the running example. We have used an Intel PC with 8 GB memory, 2.50 GHz  $\times$  4 CPU and 64-bit Ubuntu 14.04 LTS to run the experiments. The results are presented for each analyzed property.

*Example 2.* Using reachability and safety properties, and some given initial values we performed some verifications in UPPAAL for the system presented in Example 1. The system on which we performed the verification was composed out of three neurons and six automata, by using the declarations:

```
const int N = 2; //Number of synapses
typedef int[0, N - 1] id_s; //The id_s defines a vector of N integer numbers.
int n1 = 11; //Number of spikes in neuron 1
int n2 = 0; //Number of spikes in neuron 2
int n3 = 1; //Number of spikes in neuron 3
```

where “//text” represents a comment.

Since the neurons nondeterministically choose which rule to apply, the number of possible configurations of this system is high. The complexity of such systems increases even more when additional neurons and synapses are used, and that is why we use the model checker of UPPAAL for verification.

- $E \langle \rangle n1 == 2$  and  $n2 == 1$  and  $n3 == 1$  and  $output == 2$

Starting from the initial configuration, UPPAAL can be used to check if certain amounts of spikes can be obtain in the system during its evolutions. The result is shown in Fig. 11.

```
E<> n1 == 2 and n2 == 1 and n3 == 1 and output==2
Verification/kernel/elapsed time used: 0s / 0s / 0.001s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is satisfied.
```

**Fig. 11.** Verification of reachability of a given configuration

If our constructed systems is correct, we should not be able to reach configurations in which the amount of certain spikes does not respect the evolution of the model. Considering such an impossible to reach configuration:  $n1 == 2$

and  $n2 == 0$  and  $n3 == 1$  and  $output == 4$  we obtain, as expected, a negative response as shown in Fig. 12.

```
E<> n1 == 2 and n2 == 0 and n3 == 1 and output==4
Verification/kernel/elapsed time used: 0.01s / 0s / 0.014s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is not satisfied.
```

**Fig. 12.** Verification of reachability for a given configuration

- $A \langle \rangle output == i$ , for  $i \in \{1, 2, 3, 4, 5\}$

Starting from the initial configuration, UPPAAL can be used to check if certain amounts of spikes can be obtain as the output of the system. In this case we check which can be the output of the system and, depending on the applied rules, the output can be different. The results are shown in Fig. 13.

```
E<> output==1
Verification/kernel/elapsed time used: 0s / 0.01s / 0.001s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is satisfied.
E<> output==2
Verification/kernel/elapsed time used: 0s / 0s / 0.001s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is satisfied.
E<> output==3
Verification/kernel/elapsed time used: 0.01s / 0s / 0.001s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is satisfied.
E<> output==4
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is satisfied.
E<> output==5
Verification/kernel/elapsed time used: 0.01s / 0s / 0.014s.
Resident/virtual memory usage peaks: 5,684KB / 42,436KB.
Property is not satisfied.
```

**Fig. 13.** Verification that always its output is between 1 and 4

- $A[] \text{ not deadlock}$

A deadlock is a state in which no further evolution is possible. The existence of the deadlock means that the systems stops after some steps. For the above system, the result of the deadlock verification is depicted in Fig. 14.

```
A[] not deadlock
Verification/kernel/elapsed time used: 0s / 0s / 0.005s.
Resident/virtual memory usage peaks: 4,500KB / 41,348KB.
Property is not satisfied.
```

**Fig. 14.** Verification of deadlock

- $Pr[\# \leq 100](\langle \rangle output == 4)$  Estimates the probability of the *output* to be equal to 4 within 100 model time steps. The result of the verification is depicted in Fig. 15.

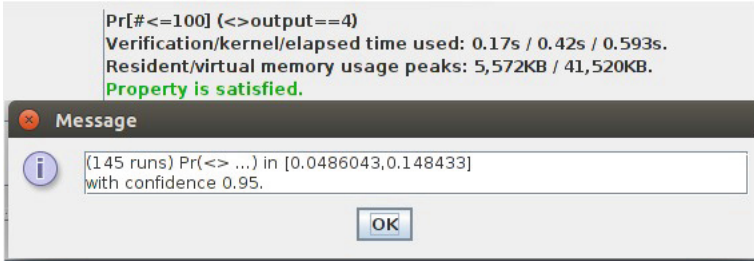


Fig. 15. Probability of reaching *output* == 4 in less than 100 steps

The tool can produce a number of histograms over model time, like probability density distribution (Fig. 16) that is useful for comparison of various distributions.

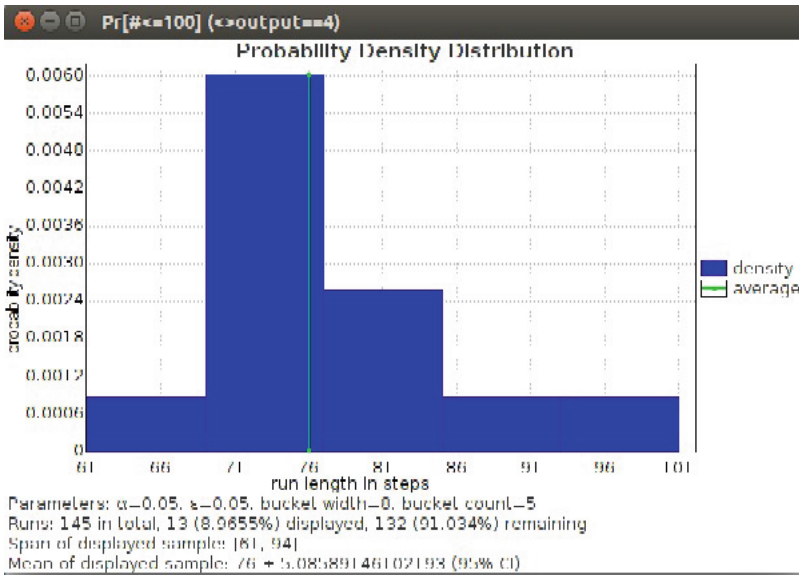


Fig. 16. Probability density distribution

## 6 Conclusion

Over the years we provided several connections between membrane systems and Petri nets for simulation and automated verification of the properties of membrane systems: enhanced mobile membranes [3, 4] are verified in [5], while mobile membrane with delays are verified in [6].

In this paper we provide a formally correct algorithm for translating stochastic spiking neural P systems into a network of timed automata, and so suitable to be verified by using UPPAAL. This allows the verification of several kinds of properties, both qualitative and quantitative, involving also the UPPAAL statistical model checking. This approach could be related to a previous attempt of modelling complex neural systems by using stochastic spiking neural P systems [8]. Due to the large number of possible reachable configurations of such a neural system, it makes sense to use various model checking capabilities of a complex software tool as UPPAAL to verify several properties: reachability of desired configurations, the fact that the system does not stop, whether the amount of resources is constant and which is the probability of some events happening.

**Acknowledgements.** Many thanks to the reviewers for their useful comments. The work was supported by a grant of the Romanian National Authority for Scientific Research, project number PN-II-ID-PCE-2011-3-0919.

## References

1. Alberts, B., Johnson, A., Lewis, J., Morgan, D., Raff, M., Roberts, K., Walter, P.: *Molecular Biology of the Cell*, 6th edn. Garland Science, New York (2014)
2. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**, 183–235 (1994)
3. Aman, B., Ciobanu, G.: Describing the immune system using enhanced mobile membranes. *Electron. Notes Theor. Comput. Sci.* **194**, 5–18 (2008)
4. Aman, B., Ciobanu, G.: Simple, enhanced and mutual mobile membranes. In: Priami, C., Back, R.-J., Petre, I. (eds.) *Transactions on Computational Systems Biology XI. LNCS*, vol. 5750, pp. 26–44. Springer, Heidelberg (2009)
5. Aman, B., Ciobanu, G.: Properties of enhanced mobile membranes via coloured Petri nets. *Inf. Process. Lett.* **112**, 243–248 (2012)
6. Aman, B., Ciobanu, G.: Verification of membrane systems with delays via Petri nets with delays. *Theor. Comput. Sci.* **598**, 87–101 (2015)
7. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. *Lect. Notes Comput. Sci.* **3098**, 87–124 (2004)
8. Cavaliere, M., Mura, I.: Experiments on the reliability of stochastic spiking neural P systems. *Nat. Comput.* **7**(4), 453–470 (2008)
9. Cerans, K.: Decidability of bisimulation equivalences for parallel timer processes. *Lect. Notes Comput. Sci.* **663**, 302–315 (1992)
10. Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J. (eds.) *Applications of Membrane Computing*. Springer, Heidelberg (2006)
11. Frisco, P., Păun, Gh., Pérez-Jiménez, M.J. (eds.) *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, Heidelberg (2014)
12. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. *Inf. Comput.* **111**, 192–224 (1994)
13. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Handbook of Membrane Computing*. Oxford University Press, New York (2010)
14. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*, vol. 3. Springer, Heidelberg (1997)
15. Xu, Z., Cavaliere, M., An, P., Vrudhula, S., Cao, Y.: The stochastic loss of spikes in spiking neural P systems: design and implementation of reliable arithmetic circuits. *Fundamenta Informaticae* **134**(1–2), 183–200 (2014)