Spiking Neural Networks as Timed Automata

Giovanni Ciatto\textsuperscript{1,2}, Elisabetta De Maria\textsuperscript{2}, and Cinzia Di Giusto\textsuperscript{2}

\textsuperscript{1} Università di Bologna, Italy
\textsuperscript{2} Université Côte d’Azur, CNRS, I3S, France

Abstract

In this paper we show how Spiking Neural Networks can be formalised using Timed Automata Networks. Neurons are modelled as timed automata waiting for inputs on a number of different channels (synapses), for a given amount of time (the accumulation period). When this period is over, the current potential value is computed taking into account the current sum of weighted inputs, and the previous decayed potential value. If the current potential overcomes a given threshold, the automaton emits a broadcast signal over its output channel, otherwise it restarts another accumulation period. After each emission, the automaton is constrained to remain inactive for a fixed refractory period after which the potential is reset. Spiking Neural Networks can be formalised as sets of automata, one for each neuron, running in parallel and sharing channels according to the structure of the network. The inputs needed to feed networks are defined through timed automata as well: we provide a language (and its encoding into timed automata) to model patterns of spikes and pauses and a way of generating unpredictable sequences.

1 Introduction

The brain behaviour has been the object of intensive studies in the last decades: on one side, researchers are interested in the inner functioning of neurons — which are its elementary components — their interactions and how such aspects participate to the ability to move, learn or remember, typical of living beings; on the other side, they emulate nature trying to reproduce such capabilities, e.g., within robot controllers, speech/text/face recognition applications, etc.

In order to achieve a detailed understanding of the brain functioning, both neurons behaviour and their interactions must be studied. Historically, interconnected neurons, “Neural Networks”, have been naturally modelled as directed weighted graphs where vertices are computational units receiving inputs by a number of ingoing edges, called synapses, elaborating and possibly propagating them over outgoing edges. Several inner models of the neuron behaviour have been proposed: some of them make neurons behave as binary threshold gates, other ones exploit a sigmoidal transfer function, while, in many cases, differential equations are employed.
According to [15, 17], three different and progressive generations of neural networks can be recognised: (i) first generation models handle discrete inputs and outputs and their computational units are threshold-based transfer functions; they include McCulloch and Pitt’s threshold gate model [16], the perceptron model [8], Hopfield networks [11], and Boltzmann machines [2]; (ii) second generation models exploit real valued activation functions, e.g., the sigmoid function, accepting and producing real values: a well known example is the multi-layer perceptron [5, 19]; (iii) third generation networks are known as Spiking Neural Networks. They extend second generation models treating time-dependent and real valued signals often composed by spike trains. Neurons may fire output spikes according to threshold-based rules which take into account input spikes magnitude and occurrence time [17].

The core of our analysis are Spiking Neural Networks. Because of the introduction of timing aspects (in particular, observe that information is represented not only by spikes magnitudes but also by their frequency), they are considered closer to the actual brain functioning than other generations models. Spiking Neural Networks are weighted directed graphs where edges represent synapses, weights serve as synaptic strengths, and vertices correspond to Spiking Neurons. The latter ones are computational units that may emit (or fire) output impulses (spikes) taking into account input impulses strength and their occurrence instants. Models of this sort are of great interest not only because they are closer to natural neural networks behaviour, but also because the temporal dimension allows to represent information according to various coding schemes [17, 18]: e.g., the amount of spikes occurred within a given time window (rate coding), the reception/absence of spikes over different synapses (binary coding), the relative order of spikes occurrences (rate rank coding) or the precise time difference between any two successive spikes (timing code). Several spiking neuron models have been proposed in literature, having different complexities and capabilities. In [13] spiking neuron models are classified according to some behaviours (i.e., typical responses to an input pattern) that they should exhibit in order to be considered biologically relevant. The Leaky Integrate & Fire (LI&F) model [14], where past inputs relevance exponentially decays with time, is one of the most studied neuron models because it is basic and easy to use [13, 17]. On the other hand, the Hodgkin-Huxley (H-H) model [10] is one of the most complete and important within the scope of computational neuroscience, being composed by four differential equations comparing neurons to electrical circuits. As one may expect, the more complex the model, the more behaviours it can reproduce, at the price of a greater computational cost for simulation and formal analysis; e.g., in [13], the H-H model can reproduce all behaviours, but the simulation process is really expensive even for just a few neurons being simulated for a small amount
Our aim is to produce a neuron model being meaningful from a biological point of view but also suitable to formal analysis and verification. We intend to exploit model checking algorithms to automatically prove whether our system verifies or not some desired properties. More precisely, this technique could be used to detect non-active portions within some network (i.e., the subset of neurons not contributing to the network outcome), to test whether a particular output sequence can be produced or not, to prove that a network may never be able to emit, to assess if a change to the network structure can alter its behaviour, or to investigate (new) learning algorithms which take time into account.

In this work, we take inspiration from the LI&F model introduced in [6], which relies on the synchronous approach based on the notion of logical time: time is considered as a sequence of logical discrete instants, and an instant is a point in time where external input events can be observed, computations can be done, and outputs can be emitted. The variant we introduce here takes into account some new time-related aspects, such a lapse of time in which the neuron is not active, i.e., it cannot receive and emit. We encode LI&F networks into Timed Automata: we show how to define the behavior of a single neuron and how to build a network of neurons.

Timed Automata [3] are Finite State Automata extended with timed behaviours: constraints are allowed limiting the amount of time an automaton can remain within a particular state, or the time interval during which a particular transition may be enabled. Timed Automata Networks are sets of automata that can synchronise over channels communications.

Our modelling of Spiking Neural Networks consists of Timed Automata Networks where each neuron is an automaton alternating between two states: it accumulates the weighted sum of inputs, provided by a number of ingoing weighted synapses, for a given amount of time, and then, if the potential accumulated during the last and previous accumulation periods overcomes a given threshold, the neuron fires an output over the outgoing synapse. Synapses are channels shared between the timed automata representing neurons, while spike emissions are represented by synchronisations occurring over such channels. Timed Automata can be exploited to produce or recognise precisely defined spike sequences, too.

The rest of the paper is organised as follows: in Section 2 we describe our reference model, the Leaky Integrate & Fire one, in Section 3 we recall definitions of Timed Automata Networks, and in Section 4 we show how Spiking Neural Networks can be encoded into Timed Automata Networks and how inputs and outputs are handled by automata. Finally, Section 5 summarises our approach and presents some future research directions.


2 Leaky Integrate and Fire Model

Spiking Neural Networks [15] are modelled as directed weighted graphs where vertices are computational units and edges represent synapses. The signals propagating over synapses are trains of impulses: spikes. Synapses may modulate such signals according to their weight or they could introduce some propagation delay. Synapses are classified according to their weight: excitatory if positive, or inhibitory if negative.

Computational units represent neurons, whose dynamics is governed by two parameters: the membrane potential (or, simply, potential) and the threshold. The former one depends on spikes received by neurons over ingoing synapses. Both current and past spikes are taken into account even if old spikes contribution is lower. In particular, the leak factor is a measure of neuron memory about past spikes. The neuron outcome is controlled by the algebraic difference between the membrane potential and the threshold: it is enabled to fire (i.e., emit an output impulse over all outgoing synapses) only if such difference is non-negative. Immediately after each emission the neuron membrane potential is reset and the neuron stays in a refractory period for a given amount of time. During this period it has no dynamics: it cannot increase its potential as any received spike is lost and therefore it cannot emit any spike.

We focus on the Leaky Integrate & Fire model that from an observational point of view is biophysically meaningful [13, 17] but is abstracted enough to be able to apply formal verification techniques such as model-checking. The original definition of Leaky Integrate & Fire traces back to [14]; here we work on an extended version of the discretized formulation proposed in [6].

Definition 1 (Spiking Integrate and Fire Neural Network). A Spiking Integrate and Fire Neural Network is a tuple \((V, A, w)\), where:

- \(V\) are Spiking Integrate and Fire Neurons,
- \(A \subseteq V \times V\) are the synapses,
- \(w : A \to \mathbb{Q} \cap [-1, 1]\) is the synapse weight function associating to each synapse \((u, v)\) a weight \(w_{u,v}\).

We distinguish three disjoint sets \(V_i\) of input neurons, \(V_{int}\) of intermediary neurons, and \(V_o\) of output neurons, with \(V = V_i \cup V_{int} \cup V_o\).

A Spiking Integrate and Fire Neuron is a tuple \((\theta_v, p_v, \tau_v, y_v)\), where:

- \(\theta_v \in \mathbb{N}\) is the firing threshold,
- \(p_v : \mathbb{N} \to \mathbb{Q}_0^+\) is the [membrane] potential function defined as

\[
p_v(t) = \sum_{i=1}^{m} w_i \cdot x_i(t) + \lambda \cdot p_v(t - 1),
\]
with \( p_v(0) = 0 \) and where \( x_i(t) \in \{0, 1\} \) is the signal received at the time \( t \) by the neuron through its \( i \)-th input synapses, and \( \lambda \in [0, 1] \) is a rational number, representing the leak factor.

- \( \tau_v \in \mathbb{N}^+ \) is the refractory period,
- \( y_v : \mathbb{N} \rightarrow \{0, 1\} \) is the neuron output function, defined as

\[
y_v(t) = \begin{cases} 
1 & \text{if } p_v(t) \geq \theta_v \\
0 & \text{otherwise}
\end{cases}
\]

As shown in the previous definition, the set of neurons of a Spiking Integrate and Fire Neural Network can be classified into input, intermediary, and output ones. The dynamics of each neuron \( v \) is defined by means of the set of its firing times \( F_v = \{t_1, t_2, \ldots\} \subset \mathbb{N} \), also called spike train. For each input neuron, the set \( F_v \) is assumed to be given as input for the network. For each output neuron, the set \( F_v \) is considered an output for the network.

### 3 Timed Automata

Timed Automata [3] are a powerful theoretical formalism for modelling and verifying real time systems. A timed automaton is an annotated directed (and connected) graph, with an initial node and provided with a finite set of non-negative real variables called clocks. Nodes (called locations) are annotated with invariants (predicates allowing to enter or stay in a location), arcs with guards, communication labels, and possibly with some variables upgrades and clock resets. Guards are conjunctions of elementary predicates of the form \( x \ op \ c \), where \( op \in \{>, \geq, =, <, \leq\} \) where \( x \) is a clock and \( c \) a (possibly parameterised) positive integer constant. As usual, the empty conjunction is interpreted as true. The set of all guards and invariant predicates will be denoted by \( G \).

**Definition 2.** A timed automaton \( TA \) is a tuple \( (L, l^0, X, \Sigma, Arcs, Inv) \), where

- \( L \) is a set of locations with \( l^0 \in L \) the initial one
- \( X \) is the set of clocks,
- \( \Sigma \) is a set of communication labels,
- \( Arcs \subseteq L \times (G \cup \Sigma \cup U) \times L \) is a set of arcs between locations with a guard in \( G \), a communication label in \( \Sigma \cup \{c\} \), and a set of variables upgrades (e.g., clock resets);
\textbullet~Inv: L \rightarrow G assigns invariants to locations.

It is possible to define a synchronised product of a set of timed automata that work and synchronise in parallel. The automata are required to have disjoint sets of locations, but may share clocks and communication labels which are used for synchronisation. We restrict communications to be broadcast through labels $b!$, $b? \in \Sigma$ meaning that a set of automata can synchronise if one is emitting; notice that, a process can always emit (e.g., $b!$) and the receivers ($b?$) must synchronise if they can.

Locations can be normal, urgent or committed. Urgent locations force the time to freeze, committed once not only freeze time but the automaton must leave the location as soon as possible, i.e., they have higher priority.

The synchronous product $TA_1 \parallel \ldots \parallel TA_n$ of timed automata, where for each $j \in [1, \ldots, n]$, $TA_j = (L_j, l^0_j, X_j, \Sigma_j, Arcs_j, Inv_j)$ and all $L_j$ are pairwise disjoint sets of locations is the timed automaton

$$TA = (L, l^0, X, \Sigma, Arcs, \text{Inv})$$

such that:

\textbullet~$L = L_1 \times \ldots \times L_n$ and $l^0 = (l^0_1, \ldots, l^0_n)$, $X = \bigcup_{j=1}^n X_j$, $\Sigma = \bigcup_{j=1}^n \Sigma_j$,

\textbullet~$\forall l = (l_1, \ldots, l_n) \in L$: \text{Inv}(l) = \bigwedge_j \text{Inv}_j(l_j)$,

\textbullet~$\text{Arcs}$ is the set of arcs $(l_1, \ldots, l_n) \xrightarrow{g,a,r} (l'_1, \ldots, l'_n)$ such that for all $1 \leq j \leq n$ then $l'_j = l_j$.

The semantics of a synchronous product $TA_1 \parallel \ldots \parallel TA_n$ is that of the underlying timed automaton $TA$ with the following notations. A location is a vector $l = (l_1, \ldots, l_n)$. We write $l[l'_j/l_j, j \in S]$ to denote the location $l$ in which the $j$th element $l_j$ is replaced by $l'_j$, for all $j$ in some set $S$. A valuation is a function $\nu$ from the set of clocks to the non-negative reals. Let $V$ be the set of all clock valuations, and $\nu_0(x) = 0$ for all $x \in X$. We shall denote by $\nu \models F$ the fact that the valuation $\nu$ satisfies (makes true) the formula $F$. If $r$ is a clock reset, we shall denote by $\nu[r]$ the valuation obtained after applying clock reset $r \subseteq X$ to $\nu$; and if $d \in \mathbb{R}_{>0}$ is a delay, $\nu + d$ is the valuation such that, for any clock $x \in X$, $(\nu + d)(x) = \nu(x) + d$.

The semantics of a synchronous product $TA_1 \parallel \ldots \parallel TA_n$ is defined as a timed transition system $(S, s_0, \rightarrow)$, where $S = (L_1 \times \ldots \times L_n) \times V$ is the set of states, $s_0 = (l^0, \nu_0)$ is the initial state, and $\rightarrow \subseteq S \times S$ is the transition relation defined by:

\textbullet~(silent): $(l, \nu) \rightarrow (l', \nu')$ if there exists $l_i \xrightarrow{g,x} l'_i$, for some $i$, such that $l' = l[l'_j/l_j], \nu \models g$ and $\nu' = \nu[r]$,
Figure 1: $TA_1$ and $TA_2$ start in the $l_1$ and $l_3$ locations, respectively, so the initial state is $[(l_1, l_3); x = 0]$. A timed transition produces a delay of 1 time unit, making the system move to state $[(l_1, l_3); x = 1]$. A broadcast transition is now enabled, making the system move to state $[(l_2, l_3); x = 0]$, broadcasting over channel $a$ and resetting the $x$ clock. Two successive timed transitions (0.5 time units) followed by a broadcast one will eventually lead the system to state $[(l_2, l_4); x = 1]$.

- **(broadcast):** $(\bar{\ell}, \nu) \to (\bar{\ell}', \nu')$ if there exist an output arc $l_j \xrightarrow{g_j, b_j, r_j} l'_j \in Arcs_j$ and a (possibly empty) set of input arcs of the form $l_k \xrightarrow{g_k, b_k, r_k} l'_k \in Arcs_k$ such that for all $k \in K = \{k_1, \ldots, k_m\} \subseteq \{l_1, \ldots, l_n\} \setminus \{l_j\}$, the size of $K$ is maximal, $\nu \models \bigwedge_{k \in K \cup \{j\}} g_k$, $\nu' = \nu[r_k, k \in K \cup \{j\}]$.

- **(timed):** $(l, \nu) \to (l, \nu + d)$ if $\nu + d \models Inv(l)$.

The valuation function $\nu$ is extended to handle a set of shared bounded integer variables: predicates concerning such variables can be part of edges guards or locations invariants, moreover variables can be updated on edges firings but they cannot be assigned to or from clocks.

In Figure 1 we exemplify timed automata usage, we consider the network of timed automata $TA_1$ and $TA_2$ with broadcast communications, and we give a possible run.

Throughout our modelling, we have used the specification and analysis tool Uppaal [4] which provides the possibility of designing and simulating Timed Automata Networks on top of the ability of testing networks against temporal logic formulae.
4 Spiking Neural Networks modelling

We present here our modelling of the Spiking Integrate and Fire Neural Network via Timed Automata Networks. Let $S = (V_i \cup V_{int} \cup V_{out}, A, w)$ be such a network (as remarked in Section 2 we distinguish between input, intermediary and output neurons). The Timed Automata Network will be obtained as the parallel composition of the encoding of each kind of neuron. More formally:

$$\parallel S = (\parallel_{N_i \in V_i} [N_i]) \parallel (\parallel_{N_j \in V_{int}} [N_j]) \parallel (\parallel_{N_o \in V_{out}} [N_o])$$

4.1 Input neurons

The behaviour of input neurons is part of the specification of the network. Here we define two kinds of input behaviours: regular and non-deterministic ones. For each family, we provide an encoding into Timed Automata.

Regular input sequences. Spike trains are “regular” sequences of spikes and pauses: spikes are instantaneous while pauses have a non-null duration. Sequences can be empty, finite or infinite. After each spike there must be a pause except when the spike is the last event of a finite sequence. Infinite sequences are composed by two parts: a finite and arbitrary prologue and an infinite and periodic part composed by a finite sequence of spike–pause pairs. More formally the input sequence $IS$ is defined by the grammar production:

$$IS ::= \varepsilon \mid \Phi \mid \Phi \Omega^\omega$$

where $\Phi$ is a finite prefix

$$\Phi ::= P? (s P)^* s$$

and $\Omega$ is the part which is repeated infinitely often

$$( (s P_1) \cdots (s P_n) )^\omega$$

with $s$ representing a spike and $P_i = p[N_i]$ is a a pause of a duration $N_i$.

It is possible to generate an emitter automaton for any regular input sequence. Such an automaton requires a clock $t$ to measure pause durations, a boolean variable $s$ which is true every time the automaton is firing and a location for each spike or pause into the sequence. The encoding $\parallel IS$ is as follows, where $\alpha$ ranges over sub-sequences:

- $\parallel \varepsilon = \{ E \}$ an empty sequence is encoded into an automaton having just one location $E$ without any edge;
Any finite sub-sequence is a list of spikes and pauses. They are recursively encoded as follows:

\[ \Phi = \text{any finite sequence is encoded into} \]

\[ \Phi \Omega^\omega = \text{any infinite sequence is composed by a finite sub-sequence } \Phi \text{ followed by a finite sub-sequence } \Omega \text{ repeated an infinite amount of times. The two sub-sequences are encoded according to the rules explained below and the resulting automata are connected. Finally, an urgent location } R \text{ is added, having an input edge from } \Omega \text{ last location and an output edge to } \Omega \text{ first location.} \]

Any finite sub-sequence is a list of spikes and pauses. They are recursively encoded as follows:

\[ J = \text{any pause having duration } N \text{ and followed by a sub-sequence } \alpha \text{ is encoded into a location } P \text{ with the invariant } t \leq T \text{ having one outgoing edge connected to the automaton } \alpha; \text{ such an edge is enabled if and only if } t = T \text{ and, if triggered, } t \text{ is reset and, since pauses are always followed by spikes, } s \text{ is set to true;} \]

\[ s \alpha = \text{any spike followed by a sub-sequence } \alpha \text{ is translated to an urgent location } S \text{ having one output edge connected to the automaton translated from } \alpha; \text{ such an edge emits on } y \text{ if triggered and resets } s. \]

**Non-deterministic input sequences.** These kinds of input sequences are useful when no assumption is available on neuron inputs. These are random sequences of spikes separated by at least \( T_{\text{min}} \) time units. Their encoding is shown in Figure 2 and the automaton behaves as follows: it waits in location \( B \) an arbitrary amount of time before moving to location \( S \), firing its first spike over channel \( x \). Since location \( S \) is urgent, the automaton instantaneously moves to location \( W \), resetting clock \( t \). Finally, from location \( W \), after an arbitrary amount of time \( t \in ] T_{\text{min}}, \infty [ \), it moves to location \( S \), firing a spike. Notice that an initial delay \( D \) may be introduced by adding invariant \( t \leq D \) to location \( B \) and guard \( t = D \) on edge \((B \rightarrow S)\).
4.2 Intermediary and Output Neurons

The neuron is designed as a synchronous and stateful machine that: i) accumulates potential whenever it receives input spikes within a given accumulation period, ii) if the accumulated potential is greater than the threshold, the neuron emits an output spike, iii) it waits during a refractory period, and restarts from i). We assume that no two input spikes on the same synapse can be received within the same accumulation period (i.e., the accumulation period is shorter than the minimum refractory period of the input neurons of the network). Next, we give the encoding of a neuron into Timed Automata. Notice that this encoding applies to intermediary and output neurons only.

Definition 3. Given an intermediary neuron \( N = (\theta, p, \tau, y) \) with \( m \) input synapses, its encoding into Timed Automata is \([N] = (L, A, X, \Sigma, Arcs, Inv)\) with:

- \( L = \{A, W, D\} \) with \( D \) committed,
- \( X = \{t\} \)
- \( \Sigma = \{x_i \mid i \in [1..m]\} \cup \{y\} \),
- \( Arcs = \{(A, t \leq T, x_i?, \{a := a + w_i\}, A) \mid i \in [1..m]\} \cup \{(A, t = T, \{p := a + \lfloor \lambda p \rfloor\}, D), (D, p < \theta, \{a := 0\}, A), (D, p \geq \theta, y!, W), (W, t = \tau, \{a := 0, t := 0, p := 0\}, A)\} \);
- \( Inv(A) = t \leq T, Inv(W) = t \leq \tau, Inv(D) = \text{true} \).

The neuron behaviour, described by the Automaton in Figure 3, depends on the following channels, variables and clocks:

- \( x_i \) for \( i \in [1..m] \) are the \( m \) input channels,
- \( y \) is the broadcast channel used to emit the output spike,
- \( p \in \mathbb{N} \) is the current potential value, initially set to 0,
\( a \in \mathbb{N} \) is the weighted sum of input spikes occurred within the current accumulation period; it is 0 at the beginning of each round.

The automaton has three locations: \( A \), \( D \) and \( W \). It can move from one location to another according to the following intuitive semantics:

- the neuron keeps waiting in state \( A \) (for Accumulation) for input spikes while \( t \leq T \) and whenever it receives a spike on input \( x_i \), it updates \( a \) with:
  \[
  a := a + w_i
  \]

- when \( t = T \), the neuron moves to state \( D \) (for Decision), resetting \( t \) and updating \( p \) according to the potential function given in Definition 1:
  \[
  p := a + \lfloor \lambda \cdot p \rfloor
  \]
  since state \( D \) is committed, it does not allow time to progress, so, from this state, the neuron can move back to state \( A \) resetting \( a \) if the potential has not reached the threshold \( p < \theta \), or it can move to state \( W \), firing an output spike, otherwise;

- the neuron remains in state \( W \) (for Wait) for \( \tau \) time units and then it moves back to state \( A \) resetting \( a \), \( p \) and \( t \).

### 4.3 Output consumers

In order to have a complete modelling of a Spiking Neural Network, for each output neuron we build an output consumer automaton \( O_y \). The automaton,
shown in Figure 4, waits in location W for the corresponding output spikes on channel y and as soon as it receives the spike, it moves to location O. This location is only needed to simplify model checking queries. Since it is urgent, the automaton instantly moves back to location W resetting $s$ the clock measuring the elapsed time since last emission and setting $e$ to its negation, with $e$ being a boolean variable which differentiates each emission from its successor.

Thus the encoding of an output neuron $N_o$ is the parallel composition of the encoding of the $N_o$ as if it was an intermediary neuron plus an output consumer on it broadcast channel $y$:

$$[[N_o]] = [[N_o]] || O_y$$

### 5 Conclusion and Future Works

In this paper we formalised the LI&F model of Spiking Neural Networks via Timed Automata Networks. LI&F neurons are modelled as automata waiting for inputs on a number of different channels, for a fixed amount of time. When such an accumulation period is over, the current potential value is computed taking into account the current sum of weighted inputs, and the previous decayed potential value. If the current potential overcomes a given threshold, the automaton emits a broadcast signal over its output channel, otherwise it restarts its accumulation period. After each emission, the automaton is constrained to remain inactive for a fixed refractory period (after which the potential is reset). Spiking Neural Networks composed by more than one neuron can be formalised by a set of automata one for each neuron, running in parallel and sharing channels accordingly. The inputs needed to feed network are defined through Timed Automata as well. We have provided a language and its encoding into Timed Automata to model patterns of spikes and pauses and a way of modelling unpredictable sequences.

We have a complete implementation of the Spiking Neural Network model proposed in the paper via the tool Uppaal. It can be found at page [1].

As for future work, we intend to validate our neuron model proving some
characteristic properties expressed in temporal logics via model-checking. Furthermore, we consider this work as the starting point for a number of research directions: we plan to study whether our model cannot reproduce behaviours requiring bursts emission capability, as stated in [13] (e.g., tonic or phasic bursting), or some notion of memory (e.g., phasic spiking, or bistability). Furthermore, it may be interesting to enrich our formalisations to include modelling of propagation delays or even more complex spiking neuron models like the theta-neuron model [7] or Izhikevich’s one [12]. Finally it may be interesting to combine learning algorithms with formal-analysis: we would like to exploit reachability properties verification to control weights variations within the scope of existing learning algorithms or strategies, e.g., Hebb’s rule [9].

References


