We propose a formalisation of spiking neural networks based on 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 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. Spiking neural networks are formalised as sets of automata, one for each neuron, running in parallel and sharing channels according to the structure of the network. The model is then validated against some crucial properties defined via proper temporal logic formulae.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS CONCEPTS
• Theory of computation → Timed and hybrid models; • Applied computing → Bioinformatics; • Computer systems organization → Neural networks;

KEYWORDS
Neural networks, Leaky Integrate and Fire Model, Timed Automata, Temporal Logic, Model Checking.

ACM Reference Format:

1 INTRODUCTION
In order to achieve a detailed understanding of the brain functioning, both neuron behaviours and their interactions became the subject of intense studies in the last decades. In the literature, “Neural Networks” have been naturally modelled as directed weighted graphs whose nodes are computational units receiving inputs from a number of ingoing edges, called synapses, elaborating and possibly propagating them over outgoing edges. Weights serve as synaptic strengths: positive weights stand for excitatory synapses and negative weights for inhibitory ones.

According to [15, 17], three different and progressive generations of neural networks can be recognised. 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] and the perceptron model [10]. 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 [7]. 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 spike magnitudes and occurrence times [17].

The core of our analysis are spiking neural networks. Because of the introduction of timing aspects, they are considered closer to the actual brain functioning than other generations models. Several spiking neuron models have been proposed in the literature, having different complexities and capabilities. In [13], Izhikevich classifies spiking neuron models according to some behaviour (i.e., typical responses to an input pattern) that they should exhibit in order to be considered biologically relevant. The leaky integrate & fire (LIF) model [14], where past inputs relevance exponentially decays with time, is one of the most studied neuron models because it is straightforward and easy to use [13, 17]. On the other end of the spectrum, the Hodgkin-Huxley (H-H) model [11] is one of the most complex being composed by four differential equations comparing neurons to electrical circuits. In [13], the H-H model can reproduce all behaviours under consideration, but the simulation process is really expensive even for just a few neurons being simulated for a small amount of time. Our aim is to produce a neuron model being meaningful from a biological point of view but also amenable to formal analysis and verification, that could be therefore 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 LIF model introduced in [8], where time is considered as a sequence of logical discrete instants. The variant we introduce here takes into account some new time-related aspects, such as a lapse of time in which the neuron is not active, i.e., it cannot receive and emit. We encode LIF networks into timed automata and a first attempt can be found in [5].
Our modelling of spiking neural networks consists of timed automata networks where each neuron is an automaton. Its behaviour consists in accumulating the weighted inputs received, for a given number of time. 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 broadcast synchronisations occurring over such channels.

As a main contribution, we analyse some intrinsic properties of the proposed model, e.g., the maximum threshold value allowing a neuron to emit, or the lack of inter-spike memory, preventing the behaviour of a neuron from being influenced by what happened before the last spike. Furthermore, we encode in temporal logics all the behaviours (or capabilities) a LI&F model should be able to reproduce according to Izhikevich and exploit model checking to prove these behaviours are reproducible in our model. Izhikevich also identifies a set of behaviours which are not expected to be reproducible by any LI&F model. We prove these limits to hold for our model, too, and we provide, for each non-reproducible behaviour, an extension of the model allowing to reproduce it.

The rest of the paper is organised as follows: in Section 2 we recall definitions of timed automata networks, temporal logics, and model checking; in Section 3 we describe our reference model, the LI&F one, and its encoding into timed automata networks; in Section 4 we study some intrinsic properties of the obtained model and in Section 5 we validate it against its ability of reproducing or not some behaviours. Finally, Section 6 summarises our contribution and presents some future research directions.

2 PRELIMINARIES

In this section we introduce the formal tools we adopt in the rest of the paper, namely timed automata and temporal logics.

Timed Automata. Timed automata [1] 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 \( op \in \{>, \geq, =, <, \leq \} \); \( x \) 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 1. A timed automaton \( TA \) is a tuple \( (L, \Sigma, X, S, Arcs, Inv) \), where

- \( L \) is a set of locations with \( \emptyset \in L \) 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 \( \Sigma \cup \{r\} \), and a set of variable upgrades (e.g., clock resets);
- \( Inv : L \rightarrow G \) 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 ones freeze time and 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 \( j \in \{1, \ldots, n\} \), \( TA_j = (L_j, I_j^0, X_j, \Sigma_j, Arcs_j, Inv_j) \) and all \( L_j \) are pairwise disjoint sets of locations, is the timed automaton

\[
TA = (L, \emptyset, X, \Sigma, Arcs, Inv)
\]

such that:

- \( L = L_1 \times \ldots \times L_n \) and \( I_0 = (I_1^0, \ldots, I_n^0) \), \( X = \bigcup_{j=1}^n X_j \), \( \Sigma = \bigcup_{j=1}^n \Sigma_j \),
- \( \forall l = (l_1, \ldots, l_n) \in L : Inv(l) = \bigwedge_{j=1}^n Inv_j(l_j) \),
- \( Arcs \) 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 the one of the underlying timed automaton \( TA \) with the following notations. A location is a vector \( l = (l_1, \ldots, l_n) \). We write \( [l_j'/l_j, j \in S] \) 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 \( v \) from the set of clocks to the non-negative reals. Let \( V \) be the set of all clock valuations, and \( v_0(x) = 0 \) for all \( x \in X \). We shall denote by \( v \in V \) the fact that the valuation \( v \) satisfies (makes true) the formula \( F \). If \( r \) is a clock reset, we shall denote by \( v[r] \) the valuation obtained after applying the clock reset \( v(R) \leq v \) and if \( d \in \mathbb{R}_{>0} \) is a delay, \( v + d \) is the valuation such that, for any clock \( x \in X \), \( v(x) + d(x) = v(x) \).

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, \( s_0 = (I_0^0, v_0) \) is the initial state, and \( \rightarrow \subseteq S \times S \) is the transition relation defined by:

- (silent): \((l, v) \rightarrow (l', v')\) if there exist \( l_i \xrightarrow{g,a,r} l_i' \) for some \( i \), such that \( l' = [l_1'/l_1, \ldots, l_n'/l_n] \), \( v = g \) and \( v' = v[r] \).
- (broadcast): \((l, v) \rightarrow (l', v')\) if there exists an output arc \( l_j \xrightarrow{g_j,b!,t_j} l_j' \in Arcs \) and a (possibly empty) set of input arcs of the form \( l_k \xrightarrow{g_k,b?,t_k} l_k' \in Arcs \), such that for all \( k \in K = \{k_1, \ldots, k_n\} \subseteq \{l_1, \ldots, l_n\} \setminus \{l_j\} \), the size of \( K \) is maximal, \( v = \bigwedge_{k \in K} \bigwedge_{j \in j} g_k \), \( l' = \biguplus_{k \in K} l_k' \in K \cup \{j\} \) and \( v' = v[r_k, k \in K \cup \{j\}] \).
- (timed): \((l, v) \rightarrow (l, v + d)\) if \( v + d \in Inv(l) \).

The valuation function \( v \) 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.
Throughout our modelling, we have used the specification and analysis tool Uppaal [3], which provides the possibility of designing and simulating timed automata networks on top of the ability of testing networks against temporal logic formulae.

**Temporal Logics and Model Checking**. Model checking is one of the most common approaches to the verification of software and hardware (distributed) systems [6]. It allows to automatically prove whether a system verifies or not a given specification. In order to apply such a technique, the system at issue should be encoded as a finite transition system and the specification should be written using propositional temporal logic. Formally, a transition system over a set $AP$ of atomic propositions is a tuple $M = (Q, T, L)$, where $Q$ is a finite set of states, $T \subseteq Q \times Q$ is a total transition relation, and $L : Q \to \mathcal{P}(AP)$ is a labelling function that maps every state into the set of atomic propositions that hold at that state.

Temporal logics are formalisms for describing the dynamical evolution of a given system. The computation tree logic CTL* allows to describe properties of computation trees. Its formulas are obtained by (repeatedly) applying Boolean connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$), path quantifiers, and atomic formulas. The path quantifier $A$ (resp., $E$) can be used to state that all the paths (resp., some path) starting from a given state have some property. The state quantifiers are $X$ (next time), which specifies that a property holds at the next state of a path, $F$ (sometimes in the future), which requires a property to hold at some state on the path, $G$ (always in the future), which imposes that a property is true at every state on the path, and $U$ (until), which holds if there is a state on the path where the second of its argument properties holds and, at every preceding state on the path, the first of its two argument properties holds. Given two formulas $\varphi_1$ and $\varphi_2$, in the rest of the paper we use the shortcut $\varphi_1 \rightarrow \varphi_2$ to denote the liveness property $AG(\varphi_1 \rightarrow AF\varphi_2)$, which can be read as "$\varphi_1$ always leads to $\varphi_2$".

The branching time logic CTL is a fragment of CTL* that allows quantification over the paths starting from a given state. Unlike CTL*, it constrains every state quantifier to be immediately preceded by a path quantifier. Given a transition system $M = (Q, T, L)$, a state $q \in Q$, and a temporal logic formula $\varphi$ expressing some desirable property of the system, the model checking problem consists of establishing whether $\varphi$ holds at $q$ or not, namely, whether $M, q \models \varphi$.

3 LEAKY INTEGRATE AND FIRE MODEL AND MAPPING TO TIMED AUTOMATA

In this section we recall the formal definition of the LIF model and its modelling via timed automata as proposed in [5].

The dynamics of neurons is governed by their membrane potential (or, simply, potential), representing the difference of electrical potential across the cell membrane. The membrane potential of each neuron depends on the spikes received over the 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 the neuron memory about past spikes. The neuron outcome is controlled by the algebraic difference between its membrane potential and its firing threshold: it is enabled to fire (i.e., emit an output impulse or spike over all outgoing synapses) only if such a difference is non-negative. Spike propagation is assumed to be instantaneous. 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.

**Definition 2 (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 synapses,
- $w : A \to \mathbb{Q} \cap [-1, 1]$ is the synapse weight function associating to each synapse $(u, v)$ a weight $\omega_{u,v}$.

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

A spiking integrate and fire neuron $v$ is characterised by a parameter tuple $(\theta_v, \tau_v, \lambda_v, p_v, y_v)$, where:

- $\theta_v \in \mathbb{N}$ is the firing threshold,
- $\tau_v \in \mathbb{N}^+$ is the refractory period,
- $\lambda_v \in \mathbb{Q} \cap [0, 1]$ is the leak factor.

The dynamics of a spiking integrate and fire neuron $v$ is given by:

\[
\begin{align*}
  p_v(t) &= \left\{ \begin{array}{ll}
  \sum_{i=1}^{m} w_i \cdot x_i(t) & \text{if } p_v(t-1) \geq \theta_v \\
  \sum_{i=1}^{m} w_i \cdot x_i(t) + \lambda_v \cdot p_v(t-1) & \text{otherwise}
  \end{array} \right. \\
  y_v(t) &= \left\{ \begin{array}{ll}
  1 & \text{if } p_v(t) \geq \theta_v \\
  0 & \text{otherwise}
  \end{array} \right.
\]

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. Each input neuron can only receive as input external signals (and not other neurons’ output). The output of each output neuron is considered as an output for the network. Output neurons are the only ones whose output is not connected to other neurons.

We present here our modelling of LI&F networks (in the following denoted as neural networks) via timed automata networks. Let $S = (V, A, w)$ be a neural network, $G$ be a set of input generator neurons (these fictitious neurons are connected to input neurons and generate input sequences for the network), and $O$ be a set of output consumer neurons (these fictitious neurons are connected to the broadcast channel of each output neuron and aim at consuming their emitted spikes). The corresponding timed automata network is obtained as the parallel composition of the encoding of input generator neurons, the neurons of the network (referred as standard neurons in the following), and output consumers neurons. More formally: $[S] = (\|_n) \in G[n_i] \| (\|_n) \in V \| (\|_n) \in O[n_i])$. The encoding of input generators and output consumers can be found in [5].
The neuron is a computational unit behaving as follows: i) it accumulates potential whenever it receives input spikes within a given accumulation period, ii) if the accumulated potential is greater than the threshold, it emits an output spike, iii) it waits during a refractory period, and restarts from i). We assume that two input spikes on the same synapse cannot 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 standard neurons into timed automata.

**Definition 3.** Given a neuron \( \nu = (\theta, \tau, \lambda, p, y) \) with \( m \) input synapses, its encoding into timed automata is

\[
\mathcal{N} = (L, A, X, \Sigma, Arcs, Inv)
\]

with:

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

where \( m \) is the number of input synapses, \( T \) is the accumulation period, and \( a \) is an integer variable storing the weighted sum of input spikes.

The automaton encoding is depicted in Figure 1. The behaviour of \( \mathcal{N} \) can be summed up as follows: 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 \( p := a + [\lambda \cdot p] \). 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. Finally, the neuron remains in state \( W \) (for Wait) for \( \tau \) time units and then it moves back to state \( A \) resetting \( a, p \) and \( t \).

In the rest of the paper, we denote with \( state_\mathcal{N}(L) \) a proposition stating if the automaton \( \mathcal{N} \) is in location \( L \); and with \( eval_\mathcal{N}(x) \) a proposition indicating the value of the clock \( x \) in \( \mathcal{N} \).

![Figure 1: Neuron model.](image)

## 4 Properties of the Model

The neuron model of Definition 3 satisfies, by construction, some properties. We can compute a minimum threshold value such that any neuron, having a threshold greater than or equal to it, will never be able to fire.

**Property 4.** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron and \( a_{\text{max}} \) be the maximum value received during each accumulation period. Then, if \( \theta \geq \frac{a_{\text{max}} - \lambda}{1-\lambda} \), the neuron is not able to fire.

**Proof.** Without loss of generality, we suppose that, during each accumulation period, \( N \) receives the maximum possible input \( a_{\text{max}} \). Then, its potential function is:

\[
p_n = a_{\text{max}} + [\lambda \cdot p_{n-1}]
\]

which is always lower than or equal to its undiscretized version:

\[
p_n \leq p_{n} = a_{\text{max}} + \lambda \cdot p_{n-1}
\]

The same inequality can be written in explicit form:

\[
p_n \leq p_{n} = \sum_{k=0}^{n} a_{n-k} \cdot \lambda^k
\]

and, since we assumed the neuron always receives \( a_{\text{max}} \), \( a_{n-k} \) is constant and does not depend on \( k \):

\[
p_n \leq a_{\text{max}} \cdot \frac{1 - \lambda^n}{1 - \lambda}
\]

This rightmost factor is a geometric series:

\[
p_n \leq a_{\text{max}} \cdot \frac{1 - \lambda^n}{1 - \lambda}
\]

which reaches its maximum value \( \frac{1}{1 - \lambda} \) for \( n \to \infty \), therefore:

\[
p_n \leq a_{\text{max}} \cdot \frac{1}{1 - \lambda}
\]

Thus, if \( \theta \geq \frac{a_{\text{max}} - \lambda}{1-\lambda} \), it is impossible for the neuron potential to reach the threshold and, consequently, the neuron cannot fire. \( \square \)

In the following, we only consider neurons that respect the previous constraint.

Apart from the minimum threshold, we can also quantify the amount of time that the neuron requires to complete an accumulate–fire–rest cycle. We show that there exists a minimum delay between neuron emissions.

**Property 5.** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron. Then the time difference between successive firings cannot be lower than \( T + \tau \).

**Proof.** Let \( A_n = \sum_{k=1}^{T} a_k + b_k \) be the sum of weighted inputs during the \( n \)-th accumulation period, then the neuron behaviour can be described as follows:

\[
p_n = A_n + [\lambda \cdot p_{n-1}]
\]

which is the potential value after the \( n \)-th accumulation period. If the neuron eventually fires an output spike, then there exists \( \hat{n} > 0 \) such that:

\[
\hat{n} = \arg\min_{n \in \mathbb{N}} (p_n : p_n \geq \theta)
\]
Formal Validation of Neural Networks as Timed Automata

I.e., the firing will occur at the end of the \( \hat{n} \)-th accumulation period, which means during the \( \hat{i} \)-th time unit since \( t_0 \), thus:

\[
\hat{i} = \hat{n} \cdot T + t_0
\]

where \( t_0 \) is the last reset time, i.e., the last instant back in time when the neuron completed its refractory period. Then the next reset time \( \hat{t'} \), i.e., the next instant in future when the neuron will complete its refractory period, after having emitted a spike, is:

\[
\hat{t'} = \hat{i} + \tau = \hat{n} \cdot T + \tau + t_0
\]

At instant \( \hat{t'} \), the neuron quits its refractory period, \( n \) is reset to 0, \( t_0 \) is set to \( t' \), and \( \hat{n} \), \( \hat{i} \) and \( \hat{t'} \) must be consequently re-computed as described above.

Such a way to describe our model dynamics allow us to express the inter-firing period as a function of \( \hat{n} \):

\[
\hat{t'} - t_0 = \hat{n} \cdot T + \tau
\]

So, the minimum inter-firing period is \( T + \tau \) for \( \hat{n} = 1 \). □

Such a property can also be verified as follows: let \( I \) be the non-deterministic input generator having \( T_{min} = 1 \) and, without loss of generality, let the initial delay \( D = T + \tau \). Then the timed automata network \( T[N]O \) satisfies the following formula:

\[
AG(\text{state}_O(O) \rightarrow \text{eval}_O(s) \geq T + \tau)
\]

where \( s \) measures the time elapsed since last firing, meaning that, whenever the output consumer receives a spike, the time elapsed since the previous received spike cannot be lower than \( T + \tau \).

The next fact states that only positive stimulations are necessary for the neuron to produce emissions.

**Fact 6.** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron, \( a(t) \) the sum of weighted inputs received during the current accumulation period, and \( p(t - 1) \) the neuron potential at the end of the previous accumulation period. If \( p(t - 1) < \theta \) and \( a(t) < 0 \), the neuron cannot fire at the end of the current accumulation period. Moreover, if \( p(t) \geq \theta \) then \( a(t) > 0 \).

The neuron potential is affected by every input spike it received since the last reset time, but every event that occurred before that instant is forgotten: i.e., neurons are memoryless.

**Definition 7 (Inter-emission memory).** Let \( N \) be a neuron, \( Z_N \) its reset times set, and \( I \) an input sequence. Then \( N \) has inter-emission memory if and only if there exist two different \( t, t' \in Z_N \) such that the output sequences produced by \( N \) as a response to \( I \) starting from \( t \) and \( t' \) are different.

**Property 8.** Neurons have not inter-emission memory.

Proof. When the neuron moves from location \( W \) to \( A \), it resets clock \( t \) and variables \( p \) and \( a \), making them equal to their initial values. This entails that the neuron, if subjected to the same input sequence, will always behave in the same way. □

**5 VALIDATION OF THE MODEL**

In this section we validate the neuron model against its ability of reproducing or not some behaviours, as described by Izhikevich in [13]. We introduce first three behaviours that are verified by our model.

**Tonic Spiking.** Tonic spiking is the behaviour of a neuron producing a periodic output sequence as a response to a persistent excitatory constant input sequence.

**Property 9 (Tonic spiking).** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron having only one ingoing excitatory synapse of weight \( w \) and let \( I \) be the input source connected to \( N \) producing a persistent input sequence. Then \( N \) produces a periodic output sequence.

The property holds by construction. It can be tested via model checking in the following way. Let \( I \) be the fixed-rate input generator having arbitrary initial delay \( D \), and let \( O \) be an output consumer. Then the timed automata network \( T[N]O \) satisfies the following formulae:

\[
\begin{align*}
\text{state}_O(O) \land \text{eval}_O(e) \sim & \text{state}_O(O) \land \neg \text{eval}_O(e) \\
\text{state}_O(O) \land \neg \text{eval}_O(e) \sim & \text{state}_O(O) \land \text{eval}_O(e)
\end{align*}
\]

where \( O \) is the location that the consumer automaton \( O \) reaches after consuming a spike and \( e \) is an alternating boolean variable whose value flips whenever \( O \) moves into location \( O \). So, whenever automaton \( O \) reaches location \( O \), it will eventually reach it again.

One may also find the value \( P \) of the period of some given neuron \( N \) by means of simulations, thus the periodic behaviour can be proven verifying the following formula:

\[
AG(\text{state}_O(O) \land \text{eval}_O(f) \rightarrow \text{eval}_O(s) = P)
\]

where \( s \) is the clock measuring the time elapsed since last spike consumed by \( O \), and \( f \) is a boolean variable of automaton \( N \) which is initially \( false \) and is set to \( true \) when edge \( (W \to A) \) fires (i.e., it indicates whether \( N \) has already emitted the first spike and waited the first refractory period or not).

**Integrator.** Integrator is the behaviour of a neuron producing an output spike whenever it receives at least a specific number of spikes from its input sources in the same accumulation period.

**Property 10 (Integrator).** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron having \( m \) synapses with maximum excitatory weight \( R \) and a threshold \( n \leq m \). Then the neuron emits if it receives a spike from at least \( n \) input sources during the same accumulation period.

As in the previous case, we can use model checking tools and test the formula stating that, if at least \( n \) generators are ready to emit (location \( S \)) while \( N \) is in \( A \), then \( O \) will eventually capture an output of \( N \):

\[
\sum_{i=1}^{m} \text{state}_I(S) \geq n \land \text{state}_N(A) \sim \text{state}_O(O)
\]

Notice that, since potential depends on past inputs too, the neuron may still be able to fire in other circumstances, e.g., if it keeps receiving less than \( n \) spikes for a sufficient number of accumulation periods, then it may eventually fire.

**Excitability.** Excitability is the behaviour of a neuron emitting sequences having a decreasing inter-firing period, i.e., an increasing output frequency, when stimulated by an increasing number of excitatory inputs.

**Property 11 (Excitability).** Let \( N = (\theta, \tau, \lambda, p, y) \) be a neuron having \( m \) excitatory synapses. Then the inter-spike period decreases as the sum of weighted input spikes increases.
Proof. If we assume the neuron is receiving an increasing number of excitatory spikes, generated by an increasing number of input sources emitting persistent inputs, then \( a_t \) is the non-negative, non-decreasing and progressing (i.e., \( \forall u \exists t : a_t > u \)) succession representing the weighted sum of inputs within the \( t \)-th time unit. Consequently, \( A_n = \sum_{k=1}^{T} a_k \) is the non-negative, non-decreasing and progressing succession counting the total sum of inputs within the \( n \)-th accumulation period. Since \( A_n \) is positive and Property 5 holds, we can prove that the inter-spike period \( t_n - t_{n-1} \) decreases. □

The following behaviours are not satisfied by the LI&F model, we show that our encoding cannot verify them as well.

**Phasic Spiking.** Phasic spiking is the behaviour of a neuron producing a single output spike when receiving a persistent and excitatory input sequence and then remaining quiescent for the rest of it. Such a behaviour depends on the neuron to have inter-emission memory.

**Property 12. Neurons cannot reproduce the phasic spiking behaviour.**

Proof. The phasic spiking behaviour requires the neuron to ignore any excitatory input spike occurring after its first emission. This means producing different outcomes, before and after the first emission, as a response to the same input sequence, which is impossible for a memoryless neuron, as stated in Property 8. □

We can extend our model to reproduce phasic spiking, see Figure 2. This variant makes the neuron able to “remember” if it is receiving a persistent excitatory input sequence. After each refractory period, the neuron moves to location \( A_H \), instead of \( A \). The only difference between \( A_H \) and \( A \) is that \( A_H \) ignores positive values of \( a_t \) at the end of each accumulation period. Conversely, a non-positive value of \( a_t \) (denoting the end of the persistent input), at the end of some accumulation period, leads the neuron back in location \( A \).

**Bursting.** A burst is a finite sequence of high frequency spikes. More formally:

**Definition 13.** A spike output sequence is a burst if it is composed by spikes having an occurrence rate greater than \( 1/(T + \tau) \), with \( \tau \) being the refractory period of the neuron.

A burst sequence is a sequence composed by bursts, subject to the following constraint: the time difference between the last spike of each burst and the first spike of the next burst is greater than \( \tau \).

**Property 14. Neurons cannot produce bursts.**

Proof. A neuron \( N \) cannot emit spikes having a rate greater than \( 1/(T + \tau) \), as stated by Property 5, so it cannot produce bursts. □

In order to reproduce bursts our model can be extended by allowing several subsequent emissions in an interval period smaller than \( \tau \). After this period all clocks and variables are reset and the accumulation-fire-rest cycle can start again.

Several bursting behaviours are described in [13]. Here we discuss only three of them, as all impossibility results depend on Property 14 and all the automata extensions are similar.

**Tonic Bursting** is the behaviour of a neuron producing a burst sequence as a response to a persistent and excitatory input sequence. Phasic Bursting is the behaviour of a neuron producing a burst as a consequence of a persistent excitatory input sequence and then remaining quiescent. Obviously the preceding behaviours require the ability of producing bursts.

**Bursting-then-Spiking** is the behaviour of a neuron producing a burst as response to a persistent excitatory input sequence and then producing a periodic output sequence. Such a behaviour, similarly to Phasic and Tonic Bursting, depends on the neuron ability of producing bursts. Moreover it requires inter-emission memory, in order to detect the beginning of a persistent sequence.

**Property 15. Neurons cannot reproduce the Tonic Bursting, Phasic Bursting and Bursting-then-Spiking behaviour.**

Proof. Follows from Property 14. □

**Spike Frequency Adaptation.** Spike Frequency Adaptation is the behaviour of a neuron producing a decreasing-frequency output sequence as a response to a persistent excitatory input sequence. In other words, the inter-emission time difference increases as the time elapses. This behaviour requires the neuron to have inter-emission memory as it should be able to keep track of the time elapsed since the beginning of the input sequence.

**Property 16. Neurons cannot reproduce the Spike Frequency Adaptation behaviour.**

Proof. The Spike Frequency Adaptation behaviour requires the neuron to detect the beginning of an excitatory input sequence and to increase the time required to fire a spike, after each emission. This means the neuron will produce different outcomes as response to equal inputs, which is impossible, as stated in Property 8. □

An extended neuron model able to reproduce Spike Frequency Adaptation behaviour is shown in Figure 3. This variant allows the refractory period to increase after each neuron emission, thus making the output frequency decrease.

**Spike Latency.** Spike Latency is the behaviour of a neuron firing delayed spikes, with respect to the instant when its potential reached or overcame the threshold. Such a delay is proportional to the strength of the signal which leads it to emission, i.e., the sum of weighted inputs received during the accumulation period preceding the emission. This behaviour requires the neuron to be able to postpone its output.
Figure 3: The extended model for Spike Frequency Adaptation behaviour. Additions are colored in blue.

Figure 5: The extended model for the Threshold variability behaviour. Additions colored are in blue.

Property 17. Neurons cannot reproduce the Spike Latency behaviour.

Proof. The property holds by construction. As location $D$ is committed, no firing can be delayed. □

An easy solution to extend our model is to introduce a delay between the instant the neuron reaches or overcomes its threshold and the actual emission instant. Such a delay $\delta$ depends on the sum of weighted inputs received during the last accumulation period. If the potential is greater than or equal to the threshold, the neuron computes the delay duration $\delta(a)$, assigning it to an integer variable $d$, and then waits in location Del for $d$ time units before emitting a spike on channel $y$. The extended version is depicted in Figure 4.

Threshold Variability. Threshold variability is the behaviour of a neuron allowing its threshold to vary according to the strength of its inputs. More precisely, an excitatory input will raise the threshold while an inhibitory input will decrease it. As a consequence, excitatory inputs may more easily lead the neuron to fire when occurring after an inhibitory input.

Property 18. Neurons cannot reproduce the Threshold Variability behaviour.

Proof. By construction the neuron threshold never changes. □

The neuron model can be extended allowing the threshold to vary after each accumulation period according to the current sum of weighted inputs (see Figure 5). The threshold variable initial value is $\theta_0$. On every firing of edge $(A \rightarrow D)$, the threshold variable is increased of $\Delta(a)$, where $a$ is the sum of weighted inputs occurred during the last accumulation period and $\Delta(a)$ is an integer value whose sign is opposite to the sign of $a$ and whose magnitude is proportional to the magnitude of $a$.

Bistability. Bistability is the behaviour of a neuron alternating between two operation modes: periodic emission and quiescence. Upon reception of a single excitatory spike, it emits a periodic output sequence and switches to a quiescent mode (no emission) as soon as it received another spike. Such a behaviour requires the neuron to (i) be able to produce a periodic output sequence, even if no excitatory spike is received, (ii) be able to remain silent when no spike is received, and (iii) be able to switch between the two operation modes upon reception of an excitatory spike.


Proof. The only possibility of obtaining a periodic output as a result of no excitatory input spike is to set $\theta = 0$. This is a limit case of Property 9. Since, by construction, the threshold cannot vary, the neuron cannot switch between the two operation modes. □

The neuron model can be modified as shown in Figure 6. This variant makes its threshold switch between 0 and a positive value at the end of any accumulation period during which it received an excitatory sum of weighted inputs $a$. A null threshold would make the neuron emit even if no input is received. Conversely, a positive threshold would prevent the neuron from emitting, if no input is received. Thus, on every firing of edge $(A \rightarrow D)$, the threshold value $\theta$ is computed by the function $bist(\cdot)$:

$$bist(\theta, a) = \begin{cases} 0 & \text{if } \theta > 0 \land a > 0 \\ \theta_0 & \text{if } \theta = 0 \land a > 0 \\ \theta & \text{if } a \leq 0. \end{cases}$$
Inhibition-induced activities. Inhibition-induced Spiking is the behaviour of a neuron producing a spike output sequence as a response to a persistent inhibitory input sequence. We thus require the neuron to be able to emit as a consequence of some inhibitory input spikes.

**Property 20.** Neurons cannot reproduce the Inhibition-induced Spiking behavior.

**Proof.** Follows from Fact 6.

An easy extension to our automata is to consider the absolute value of all inputs instead of their signed values.

Rebound activities. Rebound Spike is the behaviour of a neuron producing an output spike after it received an inhibitory input. Similarly to Inhibition-induced activities, this behaviour requires the neuron to emit as a consequence of an inhibitory input spike.

**Property 21.** Neurons cannot reproduce the Rebound Spiking behaviour.

**Proof.** Follows from Fact 6.

We can modify our encoding by setting the neuron potential to be always non-negative and by fixing the threshold to be 0 as response to an inhibitory stimulation. Recall that a null threshold would make the neuron emit even if its potential is 0. Thus, on every firing of the edge \( A \rightarrow D \), if the current sum of weighted inputs \( a \) is negative, the threshold \( \theta \) is set to 0, otherwise it is set to \( \theta > 0 \). This will allow an inhibitory stimulus to produce a rebound spike.

6 CONCLUSION

In this paper we formalised the LI&F model of spiking neural networks via timed automata networks. We have a complete implementation of the proposed model via the tool Uppaal, that can be found at the page [4].

As a main contribution, we exploited model checking to automatically validate our automaton-based mapping of the LI&F model according to a number of behaviours (i.e., typical responses to an input pattern) the LI&F model should be able to reproduce, namely tonic spiking, excitability, and integrator. Formal methods of computer science turned out to be an effective tool to validate our modelling approach.

To the best of our knowledge, there are few attempts of giving formal models for LI&F. Apart from the already discussed approach of [8], the closest related work we are aware of is [2]. In this work, the authors propose a mapping of spiking neural P systems into timed automata. The modelling is substantially different from our one. They consider neurons as static objects and the dynamics is given in terms of evolution rules while for us the dynamics is intrinsic to the modelling of the neuron. This, for instance, entails that inhibitions are not just negative weights as in our case, but are represented as forgetting rules. On top of this, the notion of time is also different: while they consider durations in terms of number of applied rules, we have an explicit notion of duration given in terms of accumulation and refractory period.

As for future work, we plan to provide analogous formalisations for more complex spiking neuron models, such as the theta-neuron model [9] or Izhikevich one [12]. We also intend to extend our model to include propagation delays, which are considered important within the scope of spiking neural networks [17]. Our extension is intended to add suitable states and clocks to model synapses.

Another interesting point to address is the relaxation of the synchronicity hypothesis. We plan to propose a different version of our timed automata modelling where we get rid of the fixed accumulation period and we allow neurons to receive spikes within a variable sliding window. Such a core modification would require to revalidate the model against the properties presented in this paper and thus to compare it to the current one.

To conclude, we intend to perform a robustness analysis of the obtained models, in order to detect which neuron parameters influence most the verification of some wished temporal properties.

REFERENCES