Modeling Third Generation Neural Networks as Timed Automata and verifying their behavior through Temporal Logic

Giovanni Ciatto, Elisabetta de Maria, Cinzia Di Giusto

Modeling Third Generation Neural Networks as Timed Automata and verifying their behavior through Temporal Logic

Giovanni Ciatto Elisabetta De Maria
Cinzia Di Giusto

February 22, 2017

Abstract

In this paper we present a novel approach to model Spiking Neural Networks. These networks, referred as third generation ones, add a new dimension to the second generation: the temporal axis. 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 properties defined via proper temporal logic formulae.

Contents

Chapter 1

Introduction

Researchers have been trying to reproduce the behavior of the brain for over half a century: on one side they are studying 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 are emulating nature trying to reproduce such capabilities e.g., within robot controllers, speech/text/face recognition applications etc.

In order to achieve a complete comprehension of the brain functioning, both neurons behavior and their interaction must be studied. Historically, interconnected neurons, "Neural Networks", have been naturally modeled as directed weighted graphs where vertexes are computational units receiving inputs by a number of ingoing arcs, called *synapses*, elaborating it, and possibly propagating it over of outgoing arcs. Several inner models of the neuron behavior have been proposed: some of them make neurons behave as binary threshold gates, other ones exploit a sigmoidal transfer function, while, in a number of cases, differential equations are employed.

According to [17, 20], three different and progressive *generations* of neural networks can be recognized: (i) first generation includes discrete and threshold based models (e.g., McCulloch and Pitt's neuron [19]); (ii) second generation consists of real valued and sigmoidal-based models, which are nowadays heavily employed in Machine Learning related tasks because of the existence of powerful learning algorithms (e.g., error back-propaga-

tion [22]); (iii) third generation, which is the focus of our work, consists of a number of model that, in addition to stimuli magnitude and differently from previous generations, take time into account.

Models from the third generation, also known as "Spiking Neural Networks", are weighted directed graphs where arcs represent synapses, weights serve as synaptic strengths, and vertexes 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 behavior, but also because the temporal dimension allows to represent information according to various *coding schemes* [20, 21]: 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). A number of spiking neuron models have been proposed in literature, having different complexities and capabilities. In [14] spiking neuron models are classified according to some *behaviors* (i.e., typical responses to an input pattern) that they should exhibit in order to be considered biologically relevant. For example the Leaky Integrate & Fire (LI&F) model [15], where past inputs relevance exponentially decays with time, is one of the most studied neuron models because of its simplicity [14, 20], while the Hodgkin-Huxley (H-H) model [11] is one of the most complex and important within the scope of computational neuroscience, being composed by four differential equations comparing the neuron to an electrical circuit. Two behaviors that every model is able to reproduce are the *tonic spiking* and *integrator*: the former one describes neurons producing a periodic output if stimulated by a persistent input, the latter one illustrates how temporally closer input spikes have a greater excitatory effect on neurons potential, making them able to act as coincidence detectors. As one may expect, the more complex the model, the more behaviors it can be reproduce, at the price of greater computational cost for simulation and formal analysis; e.g., the H-H model can reproduce all behaviors, but the simulation process is really expensive even for just a few neurons being simulated for a small amount of time [14].

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, or assess if a change to the network structure can alter its behavior; or investigate (new) learning algorithms which take time into account.

In this work, we take the discretized variant of LI&F introduced in [7] and we encode it into Timed Automata. We show how to define the behavior of a single neuron and how to build a network of neurons. Finally, we show how to verify properties of the designed system via Model Checking.

Timed Automata are Finite State Automata extended with timed behaviors: 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 interact by means of *channels*.

Our modelling of Spiking Neural Network consists of a 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 TA representing neurons, while *spike* emissions are represented by *synchronizations* occurring over such channels. Timed Automata can be exploited to produce or *recognize* precisely defined spike sequences, too.

The biophysical behaviors mentioned above are interpreted as Computational Tree Logic (CTL) formulae and are tested in `Uppaal` [3] that provides an extended modeling language for automata, a simulator for step-by-step analysis and a subset of CTL for systems verification.

The rest of the report is organized as follows: Chapter 2 exposes the theoretical background. It explains the differences between the three neural networks generations and describes our reference model, the Leaky Integrate

& Fire. It illustrates the expected behaviors. Finally it recalls definitions of Timed Automata Networks and Computational Tree Logics. Chapter 3 shows how Spiking Neural Networks are encoded into Timed Automata Networks, how inputs and outputs are handled by automata. Chapter 4 provides formal proofs for the behaviors listed above. Finally, Chapter 5 summarizes our results and presents some future research directions.

Chapter 2

Theoretical background

2.1 Neural Networks

Neural Networks are directed weighted graphs were nodes are *computational units*, also known as *neurons*, and edges represents *synapses*, i.e., connections between some neuron output and some other neuron input. Several models exist in literature and they differ on the signals that neurons emit/accept and on the way such signal are elaborated. An interesting classification has been proposed in [17] which distinguishes three different generations of Neural Networks:

1. network models within the *first generation* handle discrete inputs and outputs and their computational units are threshold-based transfer functions; this includes McCulloch and Pitt's threshold gate [19], the perceptron [9], Hopfield networks [12] and Boltzmann machines [1];

2. *second generation* models, instead, exploit real valued activation functions, e.g., the sigmoid function, accepting and producing real values: a well known example is the multi-layer perceptron [6, 22];

3. networks from the *third generation* 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 [20].

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.

We adopt Maass's definition (see [17] or [16]) because it is a widely general template which can be specialized in more fine-grained characterization by providing additional constraints. Spiking Neural Networks are modeled as directed weighted graphs where vertexes are computational units and edges represents *synapses*. The signals propagating over synapses are *trains of impulses*: *spikes*. The particular wave form of impulses must be specified by model instances. Synapses may modulate such signals according to their weight or they could introduce some propagation delay. Synapses are classified according to their weight as *excitatory*, if it is positive, or *inhibitory* if negative.

Computational units represents *neurons*, whose dynamics is governed by two variables: the *membrane potential* (or, simply, *potential*) and the *threshold*. The former one depends on spikes received by neurons over ingoing synapses, after being modulated and/or delayed. Both current and past spikes are taken into account even if old spikes contribution is lower. The latter may vary according to some rule specified by instances. 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 is reset.

Another important constraint, typical of Spiking Neural Networks, is the *refractory period*: each neuron is unable to fire for a given amount of time after each emission. Such behavior can be modeled preventing the potential to reach the threshold either by keeping the former low or the latter high.

More formally:

Definition 2.1 (Spiking Neuron)**.** A Spiking Neuron v is a tuple (θ_v, p_v, τ_v), where :

- $\theta_v : \mathbb{R}_0^+ \to \mathbb{R}$ is the *threshold* function,

- $p_v : \mathbb{R}_0^+ \to \mathbb{R}$ is the [membrane] *potential* function,
- $\tau_v \in \mathbb{R}_0^+$ is the *refractory* period.

The dynamics of some neuron v is defined by means of the set of its firing times $F_v = \{t_1, t_2, \ldots\} \subset \mathbb{R}_0^+$, also called *spike train*. Such set is defined recursively: t_{i+1} is computed as a function of the difference $p_v(t) - \theta_v(t - t_i)$:

- e.g., a simple model may simply consider the smallest t such that $p_v(t) \geq \theta_v(t - t_i)$,
- while, in a stochastic model, such difference may govern the firing probability for neuron v.

After-spike refractory behavior is achieved by making it impossible for the potential to reach and overcome the threshold. This can be modeled in two ways:

- making any neuron unable to reach the threshold, e.g., by constraining each threshold function θ_v such that: $\theta_v(t - t') = +\infty$ if $t - t' < \tau_v$ for each $t' \in F_v$;
- making any neuron ignore its inputs, e.g., by constraining each potential function p_v such that: $p_v(t - t') = 0$ if $t - t' < \tau_v$ for each $t' \in F_v$,

Next we introduce networks:

Definition 2.2 (Spiking Neural Network)**.** A Spiking Neural Network is a tuple (V, A, ε, w), where:

- V are Spiking Neurons,
- $A \subseteq V \times V$ are the synapses,
- a *response* function $\varepsilon_{u,v} : \mathbb{R}_0^+ \to \mathbb{R}$ for each synapse $(u, v) \in A$,
- a weight $w_{u,v} \in \mathbb{R}$ for each synapse $(u, v) \in A$.

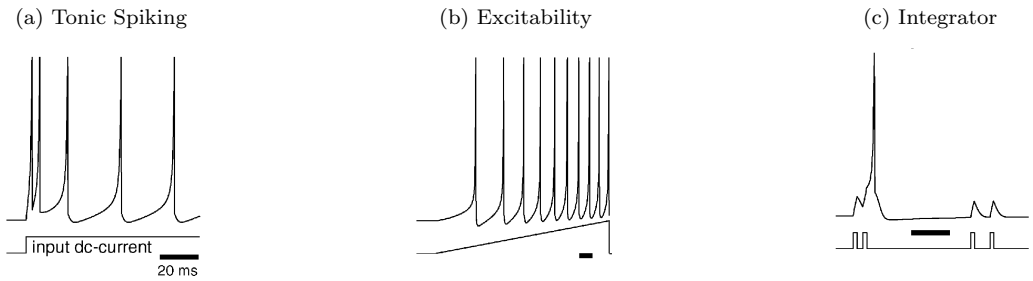


Figure 2.1: Summary and graphical representation of some of the most interesting neuron behaviors we mention within this report, taken from [14]. Each cell shows the neuron response (in the upper part) to a particular input current (in the lower part). Our interest is about 2.1a, 2.1b and 2.1c since they are the behaviors reproducible by the Leaky Integrate & Fire model.

Each response function $\varepsilon_{u,v}$ represents the impulse propagating from neuron u to neuron v and can be used to model synapse-specific features, like delays or noises.

For each neuron $v \in V - V_{in}$, the potential function p_v takes into account the response function value $\varepsilon_{u,v}(t-t')$ and the corresponding weight $w_{u,v}$, for each previous or current firing time $t' \in F_v : t' \leq t$ and for each input synapse (u, v); so the current potential may be influenced by both the current and the previous inputs. For each neuron $v \in V_{in}$, the set F_v is assumed to be given as input for the network. For all neuron $v \in V_{out}$, the set F_v is considered an output for the network.

Such definition is deliberately abstract since there exist in literature a number of models that, while respecting this definition, may differ in the way they handle e.g., potentials, signal shapes, etc.

Some authors [14,20] classify the models presented in literature according to their *biophysical plausibility*. Estimating such a feature for a given model may be a complex task since it is not well formalized. According to Izhikevich, there exists a set of *behaviors*, some shown in Figure 2.1, which a neuron may be able to reproduce. A behavior is basically a well-featured input-output relation and a model is said to be able to reproduce it if there exists at least one instance of the model presenting a comparable outcome when receiving an alike input. The author also proposes to use the amount of behaviors a model can reproduce as a measure of its biophysical plausibility.

Models	tonic spiking	phasic spiking	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accomodation	inhibition induced spiking	inhibition induced bursting
Integrate and Fire	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-
Integrate and Fire or Burst	+	+	+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	?
Quadratic Integrate And Fire	+	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-
Izhikevich's Model	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Hodgkin-Huxely	+	+	?	?	+	+	+	+	+	+	+	+	+	+	+	+	+	+	?

Figure 2.2: Comparison between several neuron models taking into account the amount of behaviors from Figure 2.1 the model can reproduce. See [14] for more detailed descriptions and for references.

As far as our work is concerned, the most interesting results are about the Integrate & Fire model capabilities. Indeeds, instances of this model should be able to reproduce the following behaviors:

tonic spiking: as a response to a persistent input, the neuron periodically fires spikes as output;

excitability: a neuron of this sort has an emission rate that linearly increases with input magnitude;

integrator: a neuron of this sort prefers high-frequency inputs: the higher the frequency the higher its firing probability; it may act as inputs coincidence detector.

2.1.1 Leaky Integrate & Fire model

Since our aim is to define a model being simple enough to be inspectable through model-checking techniques but also complex enough to be biophysically meaningful, we focused on the *Leaky Integrate & Fire*, which is one of the simplest and most studied model of biological neuron behavior (see [14] and [20]), whose original definition is traced back to [15].

We adopt the formulation proposed in [7]. It is a discretized model, amenable to formal verification, where time progresses discretely and signals are boolean-valued even if potentials are real-valued. The discretized version

is represented by the following recursive equation:

$$p_v(t) = \varepsilon_v(t) + \lambda \cdot p_v(t-1) \tag{2.1}$$

where $p_v(0)$ equals to 0 and $\lambda \in [0, 1]$ is the *leak factor*, a measure of neuron memory about past spikes.

Let m be the number of input *synapses* for some neuron v, which are modeled as boolean-valued and time-dependent *signals* $\varepsilon_i : \mathbb{N} \to \{0, 1\}$, then $\varepsilon_v(t) = \sum_{i=1}^{m} w_i \cdot \varepsilon_i(t)$ is the neuron total input. A spike *propagates* or *occurs* on the i-th synapse whenever $\varepsilon_i(t) = 1$. The neuron output is $y_v : \mathbb{N} \to \{0, 1\}$, a signal defined as follows:

$$y_v(t) = \begin{cases} 1 & if\ p_v(t) \geqslant \theta_v \\ 0 & otherwise \end{cases} \tag{2.2}$$

thus, as for inputs, an output spike occurs when $y(t) = 1$, and it is *immediately* propagated to any synapse $(v, u') \in A$ since this model does not allow delays on synapses. Finally, let t_f be the last time unit where v emitted a spike, i.e., $y_v(t_f) = 1$, then, for a given *refractory period* $\tau_v \in \mathbb{N}$, $p_v(t_f + k) = 0,\ \forall k < \tau$. Please note that during *any* refractory period:

- the neuron cannot increase its potential;
- it cannot emit any spike, since $p_v(t_f + k) < \theta_v$;
- any received spike is *lost*, i.e., it has no effect on neuron potential.

Remark. There exists an explicit version for Equation 2.1, that is:

$$p_v(t) = \sum_{k=0}^{t} \lambda^k \cdot \varepsilon_v(t-k) \tag{2.3}$$

which clearly shows how previous inputs relevance *exponentially* decays as time progresses. Such formulation is achieved as follows (subscripts are omit-

ted):

$$\begin{array}{rcccl}
p(0) & = & \varepsilon(0) & = & \lambda^0 \cdot \varepsilon(0) \\
p(1) & = & \varepsilon(1) + \lambda \cdot p(0) & = & \lambda^0 \cdot \varepsilon(1) + \lambda^1 \cdot \varepsilon(0) \\
p(2) & = & \varepsilon(2) + \lambda \cdot p(1) & = & \lambda^0 \cdot \varepsilon(2) + \lambda^1 \cdot \varepsilon(1) + \lambda^2 \cdot \varepsilon(0) \\
p(3) & = & \varepsilon(3) + \lambda \cdot p(2) & = & \lambda^0 \cdot \varepsilon(3) + \lambda^1 \cdot \varepsilon(2) + \lambda^2 \cdot \varepsilon(1) + \lambda^3 \cdot \varepsilon(0) \\
 & \vdots & & & \\
p(t) & = & \varepsilon(t) + \lambda \cdot p(t-1) & = & \sum_{k=0}^{t} \lambda^k \cdot \varepsilon(t-k)
\end{array}$$

2.2 Timed Automata

Timed Automata [2, 4] are a powerful theoretical formalism for modeling and verification of real time systems. Next, we recall their definition and semantics, their composition into Timed Automata Networks as well as the composed network semantics. We conclude with an overview on the extension introduced by the specification and analysis tool `Uppaal` [3] that we have employed here.

A Timed Automaton is a finite state machine extended with real-valued clock variables. Time progresses synchronously for all clocks, even if they can be reset independently when edges are fired. States, also called *locations*, may be enriched by *invariants*, i.e., constraints on the clock variables limiting the amount of time the automaton can remain into the constrained location. Edges are enriched too: each one may be labeled with *guards*, i.e., constraints over clocks which enable the edge when they hold, and *reset sets*, i.e., sets of clocks that must be reset to 0 when the edge is fired. Symbols, optionally consumed by edge firings, are here called *events*. More formally:

Definition 2.3 (Timed Automaton)**.** Let X be a set of symbols, each identifying one clock variable, and let G be the set of all possible guards: conjunctions of predicates having the form $x \ o \ n$ or $(x - y) \ o \ n$, where $x \in X$, $n \in \mathbb{N}$ and $o \in \{>, \geqslant, =, \leqslant, <\}$. Then a Timed Automaton is a tuple $(L, l^{(0)}, X, I, A, E)$ where:

- L is a finite set of *locations*;
- $l^{(0)} \in L$ is the initial location;

- $I : L \to G$ is a function assigning guards to locations;
- A is a set of symbols, each identifying an event;
- $E \subseteq L \times (A \cup \{\varepsilon\}) \times G \times 2^X \times L$ is a set of edges, i.e., tuples $(l,\ a,\ g,\ r,\ l')$ where:
  - l, l' are the source and destination locations, respectively,
  - a is an event,
  - g is the guard,
  - $r \subseteq X$ is the reset set.

In order to present de semantics of Timed Automata, we need to recall the definition of Labeled Transition Systems, which are a formal way to describe formal systems semantics. They consist of directed graph where vertexes are called *states*, since each of them represents a possible state of the source system, and edges are referred as *transitions*, since they represent the allowed transitions, from a state to another, for the source system. Edges are decorated through labels representing, e.g., the action firing a particular transition, the guards enabling it or some operation to be performed on their firing.

Definition 2.4 (*Labeled* Transition System)**.** Let Λ be a set of *labels*, then a Labeled Transition System is a tuple $\mathcal{M} = (S,\ s_0,\ \longrightarrow)$ where:

- S is a set of states,
- $s_0 \in S$ is the initial state,
- $\longrightarrow \subseteq S \times \Lambda \times S$ is a *transition relation*, i.e., the set of allowed transitions, having the form $s \xrightarrow{\lambda} s'$, where
  - $s, s' \in S$ are the source and destination state, respectively;
  - $\lambda \in \Lambda$ is a label.

For what concerns Timed Automaton semantics, clocks are evaluated by means of an *evaluation* function $u : X \to \mathbb{R}_0^+$ assigning a non-negative time

value to each variable in X. With an abuse of notation, we will write u meaning $\{u(x) : x \in X\}$, the set containing the current evaluation for each clock; $u + d$ meaning $\{u(x) + d : x \in X\}$, for some given $d \in \mathbb{R}_0^+$, i.e., the clock evaluation where every clock is increased of d time units respect to u. Similarly, for any reset set $r \subseteq X$, we will use the notation $[r \mapsto 0]u$ to indicate the assignment $\{x_1 \mapsto 0 : x_1 \in r\} \cup \{u(x_2) : x_2 \in X - r\}$. We will then call u_0 the function such that $u_0(x) = 0 \ \forall x \in X$ and $\mathbb{R}^X$ the set of all possible clocks evaluations. Finally, we will write $u \models I(l)$ meaning that, for some given location l, every invariant is satisfied by the current clock evaluation u.

Let $T = (L, l^{(0)}, X, I, A, E)$ be a Timed Automaton. Then, the semantics is a *labelled transition system* $(S, s_0, \rightarrow)$ where:

- $S \subseteq L \times \mathbb{R}^X$ is the set of possible states, i.e., couples (l, u) where l is a location and u an evaluation function;
- $s_0 \in S$ is the system initial state which by definition is $(l^{(0)}, u_0)$;
- $\rightarrow \subseteq S \times (\mathbb{R}_0^+ \cup A \cup \{\varepsilon\}) \times S$ is a *transition relation* whose elements can be:
  - **Delays:** modeling an automaton remaining into the same location for some period. This is possible only if the location invariants holds for the entire duration of such a period.
    Transitions of this sort share the form $(l, u) \xrightarrow{d} (l, u + d)$, for some $d \in \mathbb{R}_0^+$, and they are subjected to the following constraint: $(u + t) \models I(l), \ \forall t \in [0, d]$.
  - **Event occurrences:** modeling an automaton instantaneously moving from one location to another. This is possible only if an *enabled* edge from the source location to the destination one is defined. An edge is enabled only if its guards hold and if the destination invariants keep holding after the clocks in the edge reset set have been reset.
    Transitions of this sort share the form $(l, u) \xrightarrow{a} (l', u')$, where $a \in A \cup \{\varepsilon\}$. They are subjected to the following constraint: $\exists e = (l, a, g, r, l') \in E$ such that $u \models g$ (i.e., all guards g are

satisfied by the clock assignments u in l) and $u' = [r \mapsto 0]u$ (i.e., the new clock assignments u' are obtained by u resetting all clocks in r) and $u' \models I(l')$ (i.e., the new clock assignments u' satisfies all invariants of the destination state l').

Timed Automata Networks are a *parallel composition* of automata over a common set of clocks and communication channels obtained by means of the parallel operator $\|$. Let $\mathcal{X}$ be a set of clocks and let A_s, A_b be sets of symbols representing *synchronous* and *broadcast* communication channels respectively, such that $A_s \cap A_b = \emptyset$ and let $\mathcal{A} = \{?, !\} \times (A_s \cup A_b)$. Events in $\mathcal{A}$ are of two types:

- $?a$ is the event "sending/writing a message over/on channel a",
- $!a$ is the event "receiving/reading a message over/from channel a".

Let $N = T_1 \parallel \cdots \parallel T_n$ be a Timed Automata Network where each $T_i = (L_i, l_i^{(0)}, \mathcal{X}, I_i, \mathcal{A}, E_i)$ is a Timed Automaton. Then, its semantics is a *labelled transition system* $(S, s_0, \rightarrow)$ where:

- $S \subseteq (L_1 \times \ldots \times L_n) \times \mathbb{R}^{\mathcal{X}}$ is the set of possible states, i.e., pairs $(\mathbf{l}, u)$ where $\mathbf{l}$ is a locations *vector* and u an evaluation function;
- $s_0 \in S$ is the system initial state which by definition is $(\mathbf{l}_0, u_0)$, with $\mathbf{l}_0 = (l_1^{(0)}, \ldots, l_n^{(0)})$;
- $\rightarrow \subseteq S \times (\mathbb{R}_0^+ \cup \mathcal{A} \cup \{\varepsilon\}) \times S$ is a *transition relation* whose elements can be:
  - **Delays:** making all automata composing the network remain in respective locations for some period. This is possible only if all invariants of every automaton hold for the entire duration of such a period.
    Transitions of this sort share the form $(\mathbf{l}, u) \xrightarrow{d} (\mathbf{l}, u + d)$, for some $d \in \mathbb{R}_0^+$, and they are subjected to the following constraint: $(u + t) \models I(\mathbf{l})$ [1] $\forall t \in [0, d]$. Note that time progresses evenly for all clocks and automata.

[1] with an abuse of notation we write $I(\mathbf{l})$ instead of $I(l_1) \wedge \ldots \wedge I(l_n)$, for any $\mathbf{l} = (l_1, \ldots, l_n)$

– **Synchronous communications (synchronizations):** modeling a message exchange between two different automata. This can happen only if one of them, the *sender*, is enabled to write on some synchronous channel and the other one, the *receiver*, is enabled to read from the same channel. This means the sender must be within a location having an *enabled* outgoing edge decorated by $!a$, and, similarly, the receiver must be within a location having an *enabled* outgoing edge decorated by $?a$.
Transitions of this sort are in the form $(\mathbf{l}, u) \xrightarrow{a} (\mathbf{l}', u')$, where $a \in A_s$. They are subjected to the following constraint: there exists, for two different $i, j \in \{1, \ldots, n\}$, two edges $e_i = (l_i, !a, g_i, r_i, l_i')$ and $e_j = (l_j, ?a, g_j, r_j, l_j')$ in $E_1 \cup \cdots \cup E_n$ such that $u \models (g_i \wedge g_j)$ and $u' = [(r_i \cup r_j) \mapsto 0]u$ and $u' \models I(\mathbf{l}')$, where $\mathbf{l}' = [l_i \mapsto l_i', l_j \mapsto l_j']\mathbf{l}$; so a synchronous communication makes two automata fire their edges e_i and e_j atomically. If more that a couple of automata can synchronize, one will be chosen non-deterministically.

– **Broadcast communications:** modeling a message spreading over some channel from a sender automaton to any automaton interested in receiving messages from that channel. The main difference from synchronizations is that, here, senders can write their message even if no one is ready to receive it: thus senders cannot get stuck and massages can be lost. This transition is possible only if the sender is enabled to write on some broadcast channel a. The set of receiving automata is computed taking into account the ones being within a location having an *enabled* outgoing edge decorated by $?a$. This set must then be filtered, removing those automata which would move to a location whose invariants would be violated by some clock reset caused by this transition.
More formally, transitions of this sort share the form $(\mathbf{l}, u) \xrightarrow{a} (\mathbf{l}', u')$, where $a \in A_b$. They are subject to the following constraint: there exists in $E_1 \cup \cdots \cup E_n$
  - an edge $e_i = (l_i, !a, g_i, r_i, l_i')$, for some $i \in \{1, \ldots, n\}$,
  - a subset D' containing *all* edges having the form $e_j = (l_j, ?a, g_j, r_j, l_j')$

such that $u \models (g_j)$, where $i \neq j \in \{1, \ldots, n\}$,

- a subset $D = D' - \{e_t \in D' : u'' \models I(\mathbf{l}')\}$, where $u'' = [(r_t) \mapsto 0, \forall t : e_t \in D' \cup \{e_i\}]u$,

thus, for each e_k in $D \cup \{e_i\}$, $u \models (g_k)$ and $u' = [(r_k) \mapsto 0, \forall k]u$ and $u' \models I(\mathbf{l}')$, where $\mathbf{l}' = [l_k \mapsto l'_k, \forall k]\mathbf{l}$. So a broadcast communication make a number of edges fire atomically and it only requires an automaton to be enabled to write. If more than one broadcast communication can occur, one is chosen non-deterministically.

– **Moves:** modeling an automaton unconstrained movement from a location to another because of an edge firing. This requires the edge guards to hold within the source state and the destination location invariants to hold after clock resets have been performed. Such transitions have the form $(\mathbf{l}, u) \xrightarrow{\varepsilon} (\mathbf{l}', u')$, are subject to the following constraint: $\exists e = (l, \varepsilon, g, r, l')$ in $E_1 \cup \cdots \cup E_n$ such that $u \models g$ and $u' = [r \mapsto 0]u$ and $u' \models I(\mathbf{l}')$.

To simulate and verify our systems we use `Uppaal` [3]. It provides some extensions that we describe informally:

- a set of shared *bounded integer variables* is part of the states of the transition system defining Timed Automata Networks semantics: 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/from clocks;

- locations can be marked as *urgent* meaning that time cannot progress until an automaton remains in such a location: it is semantically equivalent to a locations labeled by the invariant $x \leq 0$ for some clock x where all ingoing edges reset x;

- locations can also be marked as *committed* meaning that, as for urgent locations, they do not allow the time to progress and they constrain any outgoing or ingoing edge to be fired *before* any edge not involving committed locations. If more than one edge involving committed locations can fire, then one is chosen non-deterministically.

`Uppaal` modeling language actually includes other features that were exploited. For a more detailed description consider reading [3].

When representing Timed Automata edges we will indicate three sections `G`, `S` and `U` respectively containing `G`uards, communications/`S`ynchornizations and `U`pdates list, where an update can be a clock reset and/or a variable assignment.

2.3 Formal Verification

This section aims at introducing a number of concepts which are useful to express the expected behavior of Spiking Neural Networks and automatically verify it. We recall definitions for CTL and the *Model Checking* problem [5] and, finally, we discuss about the `Uppaal` model checker.

Temporal logics are extensions of the first order logic allowing to represent and reason about temporal properties of some given formal system.

In this report we use CTL to express properties of the systems.

Definition 2.5 (CTL Syntax)**.** Let P be the variable ranging over atomic propositions, then a CTL formula ϕ is defined by:

$$\begin{aligned} \phi \;&=\; P \mid \textbf{true} \mid \textbf{false} && \text{atoms} \\ &\quad \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \implies \phi \mid \phi \iff \phi && \text{connectives} \\ &\quad \mid A\psi \mid E\psi && \text{path quantifiers} \\ \\ \psi \;&=\; X\phi \mid F\phi \mid G\phi \mid \phi U \phi && \text{state quantifiers} \end{aligned}$$

where $\neg$, $\wedge$, $\vee$, $\implies$ and $\iff$ are the usual logic connectives, A and E are *path quantifiers* and X, F, G and U are path-specific *state quantifiers*.

CTL formulae can only contain *couples* of quantifiers, here we give an intuition of their semantics. A formal definition can be fount in [5].

$AG\phi$ – **Always:** ϕ holds in every reachable state

$AF\phi$ – **Eventually:** ϕ will eventually hold at least in one state on every reachable path

$AX\phi$ – **Necessarily Next:** ϕ will hold in every successor state

$A(\phi_1 U \phi_2)$ – **Necessarily Until:** in every reachable path, ϕ_2 will eventually hold and ϕ_1 holds while ϕ_2 is not holding

$EG\phi$ – **Potentially always:** there exists at least one reachable path where ϕ holds in every state

$EF\phi$ – **Possibly:** there exists at least one reachable path where ϕ will eventually hold at least once

$EX\phi$ – **Possibly Next:** there exists at least one successor state where ϕ will hold

$E(\phi_1 U \phi_2)$ – **Possibly Until:** there exist at least one reachable path where ϕ_2 will eventually hold and ϕ_1 holds while ϕ_2 is not holding

The formula $AG(\phi_1 \implies AF\phi_2)$ is a common pattern used to express *liveness properties*, i.e., desirable events which will eventually occur. The formula can be read as: "ϕ_1 always leads to ϕ_2" or "whenever ϕ_1 is satisfied, then ϕ_2 will eventually be satisfied". Formulae of this sort are sometimes written using the alternative notation $\phi_1 \rightsquigarrow \phi_2$.

Model-checking is an approach to system verification aiming to test whether a given temporal logic formula holds for a given formal system, starting from a given point in time. It generally assumes that a transition system can be build, somehow representing all possible states and all allowed transitions for the given system. The verification process usually consists into exhausting all reachable states from a given *initial state*, searching for a violation of the property. If none is found, then the property is satisfied, otherwise a counter-example, also known as *trace*, is returned, i.e., a path from the initial state to the state violating the property.

In order to test some formulae we use the `Uppaal` model checker: it

employs a subset of CTL defined as follow:

$$\begin{array}{rcll} \phi & = & AG\psi \mid AF\psi \mid EG\psi \mid EF\psi & \text{quantifiers} \\ & & \mid \psi \rightsquigarrow \psi & \text{leads-to} \\ \\ \psi & = & \mathbf{true} \mid \mathbf{false} \mid \mathbf{deadlock} \mid P & \text{atoms} \\ & & \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \implies \psi & \text{connectives} \end{array}$$

where P, as usual, ranges over atomic propositions and **deadlock** is an atomic proposition which holds only in states having no outgoing transition.

Chapter 3

Spiking Neural Networks formalization

This chapter describes how to encode Spiking Neural Networks into Timed Automata Networks. We begin by showing several ways to represent LI&F neurons.

3.1 Leaky Integrate & Fire neurons as Timed Automata

Since our tool `Uppaal` does not allow to use real numbers, which would be a desirable capability when handling synapses weights and leak factors, we decided to:

- discretize the [0, 1] interval splitting it into R parts, where R is a positive integer referred as *discretization granularity*,
- represent the leak factor as a rational number.

3.1.1 Asynchronous Model

The first way, referred as *Asynchronous Model*, does not explicitly exploit the concept of *time-quantum*. It assumes that: (i) time is continuous; (ii) two input spikes cannot occur at the same instant.

```
// Type definition for rational numbers
typedef struct {
    int num;
    int den;
} ratio_t;

// Discretization granularity
const int R = <int>;

// Type definition for weights
typedef int[-R, R] weight_t;

const int M = <int>; // Number of inputs
```

Listing 3.1: `Uppaal` global definitions common to all models: the `R` constant represent the discretization granularity which is used to discretized the unitary interval, in deeds the `weight_t` type contains integers from `-R` to `+R`; while the `ratio_t` type contains rational numbers and is used for leak factors.

Definition 3.1 (Asynchronous Neuron)**.** Let $m \in \mathbb{N}$, then a neuron $\mathcal{N}_A$ is a tuple $(\mathbf{w}, \lambda, \theta, \tau)$, where:

- $\mathbf{w} = (w_1, \ldots, w_i, \ldots, w_m) \in \{-R, \ldots, R\}^m$ is the m-uple of weights,
- $\lambda : \mathbb{R}_0^+ \to [0, 1]$ is the leak factor *function* which must be a decreasing function of time used to calculate $\lambda(t - t')$, i.e., the potential leak between the current spike t and the previous one t',
- $\theta \in \mathbb{N}$ is the threshold, which is a natural number whose magnitude must be considered related to R like for any w_i,
- $\tau \in \mathbb{N}^+$ is the refractory period.

The main obstacle to such a definition is the fact that `Uppaal` (as of version `4.1.19`) does not allow to compute time depending functions, which means $\lambda(t)$ cannot be calculated. In order to work around such a limitation, we add the following assumption:

> consecutive input spikes will occur with an *almost constant* frequency regardless of which synapsis they come from, i.e. the time difference between one spike and its successor is considered to be *the same*

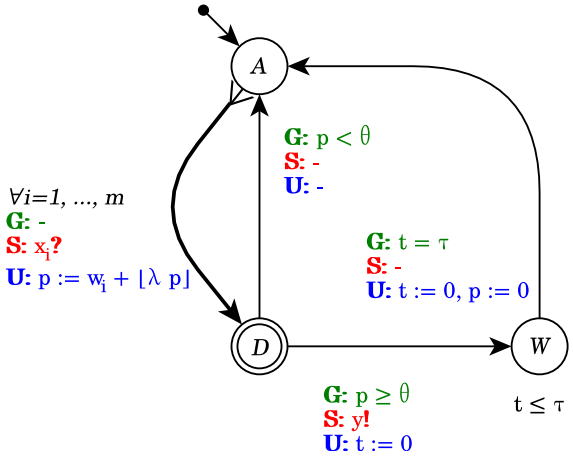


Figure 3.1: *"Asynchronous neuron"* model. Please note the graph represents a Timed Automaton, it does not represent a Neural Network. The initial state is **A**ccumulate, **D**ecide is a *committed* state while **W**ait is a normal state subject to the $t \leqslant \tau$. The $(A \to D)$ edge is actually a parametric and synthetic way to represent m edges, one for each input synapsis.

so it is possible to consider the *leak factor* as a constant instead of a decreasing function of time, leading to the following refined definition:

Definition 3.2. Let $m \in \mathbb{N}$, then an Asynchronous Neuron $\mathcal{N}_A$ is a tuple $(\mathbf{w}, \lambda, \theta, \tau)$, where:

- $\mathbf{w} = (w_1, \ldots, w_m) \in \{-R, \ldots, R\}^m$ is the m-uple of weights,
- $\lambda \in \mathbb{Q} \cap [0, 1]$ is the leak factor,
- $\theta \in \mathbb{N}$ is the threshold,
- $\tau \in \mathbb{N}^+$ is the refractory period.

The neuron behavior is then described by the Timed Automaton in Figure 3.1 and depends on the following channels, variables and clocks:

- $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_m)$ is the m-uple of broadcast channels used to receive input spikes,
- y is the output broadcast channel used to emit the output spike,
- $p \in \mathbb{N}$ is an *integer variable* holding the current potential value, which is initially 0,

- $t \in \mathbb{N}$ is a clock, initially set to 0.

The automaton has three locations: **A**, **D** and **W**, which respectively stand for **A**ccumulate, **D**ecide and **W**ait. It can move from one location to another according following rules:

- it keeps waiting in location **A** for input spikes and whenever it receives a spike on input x_i (i.e. it *receives* on channel x_i) it moves to location **D** updating p as follows:

$$p := w_i + \lfloor \lambda \cdot p \rfloor$$

- while the neuron is in location **D** then time does not progress (since it is *committed*); from this location, the neuron moves back to **A** if $p < \theta$, or it moves to **W**, firing an output spike (i.e. *writing* on y) and resetting t, otherwise;

- the neuron will remain in location **W** for an amount of time equal to τ and then it will move back to location **A** resetting both p and t.

Remark. The assumptions this model relies on are maybe too strong: it does not handle properly scenarios having input spikes occurrence times with non-negligible variance and it is expected to behave poorly in such cases. Basically, if no input spike occurs, time flow has no effect on the neuron, which is far from truth.

Implementation through `Uppaal`. A neural network with asynchronous neurons is implemented as an `Uppaal` system having global definitions shown in Listing 3.1. Each neuron is realized as a *Template* having the following parameters list:

```
// One broadcast chan &xi for each input
broadcast chan &x1, ... , weight_t &w[M_<name>], broadcast
    chan &y
```

and declarations shown in Listing 3.2. Procedure `input(i)` is executed on each (**A** $\rightarrow$ **D**) edge firing, while `reset()` is executed on each firing of edge (**W** $\rightarrow$ **A**).

```
clock t = 0;

int tau = <int>;
int theta = <int>;
ratio_t lambda = { <int>, <int> };

int p = 0;

void input(int i) {
    p = (w[i] * lambda.den + lambda.num * potential) / lambda
    .den;
}

void reset() {
    p = 0;
}
```

Listing 3.2: Asynchronous neuron *template declarations* in `Uppaal`

Finally, it may be noticed that a minimal automaton can be obtained collapsing locations **A** and **D**. The reasons they have been kept separated are: (a) within some *model-checking* query, the presence of location **D** allows to express concepts like *"the neuron has received a spike"* or *"the neuron is going to emit"*; (b) when actually implementing the neuron in `Uppaal`, the presence of location **D** allows to reduce the number of required edges: without **D** we would have needed m loops on location **A** and m edges from **A** to **W**, so $2m + 1$ total edges, considering the one from **W** to **A**; while thanks to **D** we only need $m + 3$ edges.

3.1.2 (Partially) Synchronous Model

We present here a second approach aimed at overcoming the limitations of the asynchronous model introduced above: it handles input spike co-occurrence, and time-dependent potential decay, even if no spike is received. The neuron is conceived 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

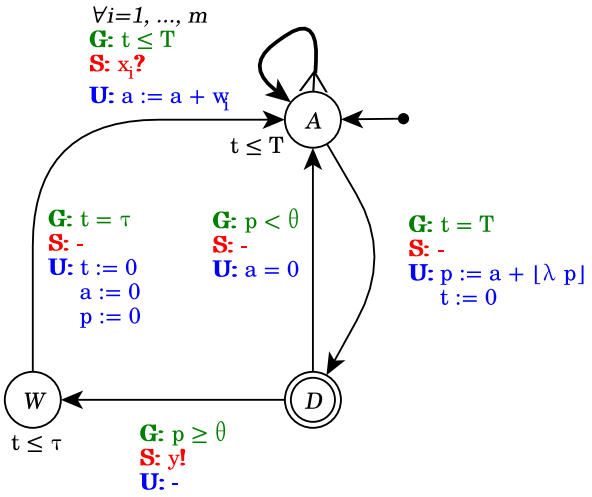


Figure 3.2: *"Synchronous neuron"* model. The $(A \to A)$ loop is actually a parametric and synthetic way to represent m edges, one for each input synapsis.

emits an output spike, (iii) it waits for *refractory period*, (iv) and *resets* to initial state. 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).

Definition 3.3 (Synchronous Neuron)**.** Let $m \in \mathbb{N}$, then a Synchronous Neuron $\mathcal{N}_S$ is a tuple $(\mathbf{w}, T, \lambda, \theta, \tau)$, where:

- $\mathbf{w} = (w_1, \ldots, w_m) \in \{-R, \ldots, R\}^m$ is the m-uple of weights,
- $\lambda \in \mathbb{Q} \cap [0, 1]$ is the leak factor,
- $\theta \in \mathbb{N}$ is the threshold,
- $\tau \in \mathbb{N}^+$ is the refractory period,
- $T \in \mathbb{N}^+$ is the accumulation period.

The neuron behavior, described by the Timed Automaton shown in Figure 3.2, depends on the following channels, variables and clocks:

- t, $\mathbf{x}$, y and p are, respectively, a clock, the m-uple of input broadcast channels, the output broadcast channel and the current potential variable, as for the asynchronous model,

- $a \in \mathbb{N}$ is a variable holding the weighted sum of input spikes occurred within the *current* accumulation period; it is 0 at the beginning of each period.

Locations are named as in the asynchronous model, but this one is subjected to different rules:

- the neuron keeps waiting in state **A** for input spikes while $t \leqslant T$ and whenever it receives a spike on input x_i it updates a as follows:

$$a := a + w_i$$

- when $t = T$ the neuron moves to state **D**, resetting t and updating p as follows:

$$p := a + \lfloor \lambda \cdot p \rfloor$$

- since state **D** is *committed*, it does not allow time to progress, so, from this state, the neuron can move back to **A** resetting a if $p < \theta$, or it can move to **W**, firing an output spike, otherwise;

- the neuron will remain in state **W** for τ time units and then it will move back to state **A** resetting a, p and t.

The innovation here is the concept of accumulation period. According to the asynchronous model, two inputs cannot occur into the same instant and, above all, their relative order is the only thing that influences the neuron potential: two consecutive input spikes would have the same effect regardless of their time difference. Thanks to the accumulation period of the synchronous model, the time distance between two consecutive spikes can be valorized: since the $(\mathbf{A} \to \mathbf{D})$ edge firing, namely "the end of the accumulation period", is *not* governed by input spikes as in the asynchronous model but only by time, the neuron potential will actually decay as time progress if no input is received.

Note that, if the assumption requiring one input not to emit more than once within the same accumulation period does not hold (i.e. inputs frequencies are *too high*), the neuron potential would increase as if the two spikes were from different synapses.

```
const int T = <int>;
clock t = 0.0;
const int tau = <int>;
const int theta = <int>;
ratio_t lambda = { <int>, <int> };

int a = 0;
int p = 0;

void reset() {
    a = 0;
    p = 0;
}

void input(int i) {
    a += w[i];
}

void endAccumulation() {
    p = (a * lambda.den + p * lambda.num) / lambda.den;
}
```

Listing 3.3: Synchronous neuron *template declarations* in `Uppaal`

Implementation through `Uppaal`. A neural network with synchronous neurons is implemented as an `Uppaal` system having global definitions shown in Listing 3.1. It differs from the asynchronous model implementation only for the declarations, as shown in Listing 3.3. Procedure `input(i)` is executed on each ($\mathbf{A} \to \mathbf{A}$) loop firing, `endAccumulation()` is invoked on ($\mathbf{A} \to \mathbf{D}$), while `reset()` is executed on each firing of edge ($\mathbf{W} \to \mathbf{A}$). As for the asynchronous model, a minimal automaton can be obtained by removing state **D** and adding more edges.

3.2 Spiking neural networks as Timed Automata Networks

After showing how a Timed Automaton can represent a neuron, the main concern is about neuron interconnection, i.e., representing a Neural Network as a Timed Automata Network by means of some proper channel sharing convention. Another relevant matter covered by this section is about inputs and outputs representation, analysis and governance.

In order to make our models easier to inspect, we defined a language for *input sequences* specification. Here we show how to translate any word from such a language into a Timed Automaton able to emit it. Then we introduce *non-deterministic input generators* which are useful in those contexts where neurons must handle generic input sequences. Finally, we show how *output consumers* can be used to measure a neuron spike frequency.

Synapses connecting neurons are represented by automata sharing channels. More formally, let $\mathcal{I}_1, \mathcal{I}_2, \ldots$ be input generators, let $\mathcal{N}, \mathcal{N}_1, \mathcal{N}_2, \ldots$ be neurons, and let $\mathcal{O}$ be an output consumer; then synapses are Timed Automata Networks obtained by parallel composition as follows:

- input generators to neuron: $(\mathcal{I}_1, \ldots, \mathcal{I}_n) \overset{\mathbf{x}}{\parallel} \mathcal{N}$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and each x_i is a channel shared by $\mathcal{I}_i$ and $\mathcal{N}$, carrying input spikes from the former to the latter;
- neurons to neuron: $(\mathcal{N}_1, \ldots, \mathcal{N}_n) \overset{\mathbf{y}}{\parallel} \mathcal{N}$, where $\mathbf{y} = (y_1, \ldots, y_n)$ and each y_i is a channel shared by $\mathcal{N}_i$ and $\mathcal{N}$, carrying $\mathcal{N}_i$ outputs which

are received by $\mathcal{N}$;

- neuron to output consumer: $\mathcal{N} \overset{y}{\parallel} \mathcal{O}$, where y is a shared channel carrying $\mathcal{N}$ outputs which are consumed by $\mathcal{O}$.

3.2.1 Input generators

Regular input generators. Essentially, input sequences are sequences of spikes and pauses: spikes are instantaneous while pauses have a non-null duration. Sequences can be *empty*, *finite* of *infinite*. After each spike there must be a pause except when the spike is the last event of a finite sequence, i.e., there exists no sequence having two consecutive spikes. Infinite sequences are composed by two parts: a finite and arbitrary prologue and an infinite and periodic part whose period is composed by a finite sequence of *spike–pause* couples.

Definition 3.4 (Input Sequence Grammar)**.** Let s, p,] and [be terminal symbols, let I, N, P_1, ..., P_n and P be non-terminals and let $x_1, \ldots, x_n \in \mathbb{N}^+$ be some *durations* for a given $n > 0$, then:

$$\begin{aligned} I &::= \varepsilon \mid P?\,(s\,P)^*\,(s\,\varepsilon \mid ((s\,P_1) \cdots (s\,P_n))^\omega) \\ P &::= p[N] \\ p_1 &::= p[x_1] \\ &\vdots \\ P_n &::= p[x_n] \end{aligned}$$

represents the *ω-regular expression* for valid input sequences.

In Definition 3.4:

- s is a symbol representing a spike;
- p is a symbol representing a pause;
- according to the productions of P and P_i, each pause is associated to a natural-valued duration;
- $p[N]$ represents a pause whose duration is *some* number matching N, the regular expression for natural numbers;

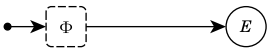
- $p[x_i]$ represents a pause whose duration is *a given number* x_i.

Notice that any pause within any valid input sequence is followed by a spike. We denote with Φ the finite prefix of an input sequence and with Ω the part which is repeated infinitely often, while α ranges over sub-sequences.

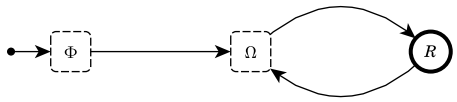
It is possible to generate an emitter automaton for any valid input sequence. Such an automaton requires a clock t to measure pauses durations, a boolean variable s which is *true* every time the automaton is firing and a location for each spike or pause into the sequence. The encoding $[\![\ I\ ]\!]$ of a sequence $I = \varepsilon \mid \Phi \mid \Phi\,\Omega^\omega$ is as follows:

- $[\![\ \varepsilon\ ]\!] =$ 

an *empty* sequence is encoded into an automaton having just one location **E** without any edge;

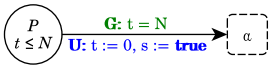
- $[\![\ \Phi\ ]\!] =$ 

any *finite* sequence is encoded into a sequence of locations, as described below, where the last one has no outgoing edges and represent the end of the sequence;

- $[\![\ \Phi\,\Omega^\omega\ ]\!] =$ 

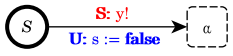
any *infinite* sequence is composed by a finite sub-sequence Φ followed by a finite sub-sequence Ω repeated an infinite amount of times. The two sub-sequences are encoded according to the rules explained below and the resulting automata are connected. Finally, an urgent location **R** is added, having an input edge from Ω last location and an output edge to Ω first location.

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

- $[\![\ p[N]\,\alpha\ ]\!] =$ 

any pause having duration N and followed by a sub-sequence α is encoded into a location **P** with the invariant $t \leqslant T$ having one outgoing edge connected to the automaton $[\![\ \alpha\ ]\!]$; such an edge is enabled if and only if $t = T$ and, if triggered, t is

reset and, since pauses are always followed by spikes, the s variable is set to *true*;

- $[\![\ s\,\alpha\ ]\!] =$ 

any spike followed by a sub-sequence α is translated to an *urgent* location **S** having one output edge connected to the automaton translated from α; such an edge emits on y if triggered and resets s.

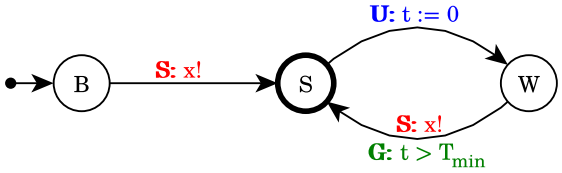
Non-deterministic input generators. If no assumptions are available or desirable about some neuron inputs, one can exploit non-deterministic input generators, i.e., automata able to fire randomly and only constrained to wait an amount of time T_{min} between an emission and its successor. An automaton of this sort is shown in Figure 3.3a and 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,
- from location **W**, after an arbitrary amount of time $t \in\ ]\,T_{min},\ \infty\,[$, it moves to location **S**, firing a spike.

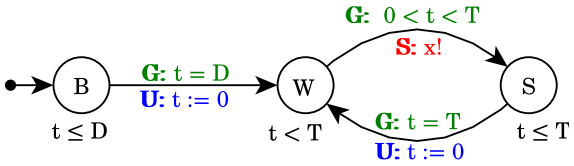
Remark. One may introduce an initial delay D by adding invariant $t \leq D$ to location **B** and guard $t = D$ on edge $(\mathbf{B} \rightarrow \mathbf{S})$

Fixed-rate input generators. Some contexts may consider input sequences having fixed *rates*, i.e., the expected amount of spikes during some given time window T is constant, even if the sequence is not formally periodic since the distribution of spikes within two different time windows may differ. We propose the automaton shown in Figure 3.3b, which is able to non-deterministically produce an output spike for each discrete time window T, after it has been quiescent for an initial delay D:

- it waits in location **B** until clock t value equals to D, then it moves to location **W**, resetting it;



(a) "*Non-deterministic* input generator"



(b) "*Fixed-rate* input generator"

Figure 3.3: Automata generating input sequences. Non-deterministic generators are only constrained to wait more than T_{min} time units between emissions. Fixed-rate generators are only constrained to fire exactly once for each period T.

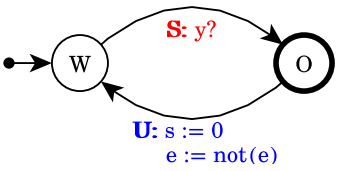


Figure 3.4: *"Output consumer"* automaton. Its initial location is **W**ait, location **O**utput is urgent, **s**ince-last-spike is a clock while **e**ven is a boolean variable.

- it waits in location **W** a non-deterministic amount of time $t_s \in]\,0,\, T\,[$ and then it moves to location **S** firing a spike over channel x;
- finally, it waits $T - t_s$ time units in **S** before moving back to **W**.

3.2.2 Output consumers

As shown in Section 3.1, neuron models emit outputs writing on some broadcast channel y. In order to query a model-checker about output neurons outcomes, the *output consumer* automaton shown in Figure 3.4 is connected to each neuron by sharing its output channel. Its behavior is straightforward:

- it waits in location **W** for the neuron it is connected to to emit an output spike, which makes it move to location **O**;
- since location **O** is urgent, the automaton will instantly move back to location **W** resetting s and setting e to its negation;

where s is the clock measuring the elapsed time since last emission and e is a *boolean variable* which differentiates each emission from its successor.

So if an output consumer automaton is in location **O** then its corresponding neuron has just emitted one spike.

Chapter 4

Validation of the Leaky Integrate & Fire model

In this chapter we validate the synchronous neuron model against its ability of reproducing some behaviors, as described by Izhikevich in [14].

First we recall the following concepts:

Definition 4.1 (Input (sub-)sequence)**.** Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be a *input sources* (i.e., neuron or generators) connected to some neuron $\mathcal{N}$, and let $F_i = \{t_{i,1}, t_{i,2}, \ldots\}$ be the *ordered* set of firing times of $\mathcal{I}_i$; then $I = \bigcup_{i=1}^{m} F_i$ is the *ordered* input sequence of $\mathcal{N}$. For any *continuous* interval $Q \subset \mathbb{R}_0^+$ the set $I \cap Q$ is a sub-sequence of I.

Definition 4.2 (Output (sub-)sequence)**.** Let $\mathcal{N}$ be a neuron and let $F_{\mathcal{N}} = \{t_1, t_2, \ldots\}$ be its *ordered* set of firing times; then $F_{\mathcal{N}}$ is the *ordered* output sequence of $\mathcal{N}$. For any *continuous* interval $Q \subset \mathbb{R}_0^+$ the set $F_{\mathcal{N}} \cap Q$ is a sub-sequence of $F_{\mathcal{N}}$.

Definition 4.3 (Persistent input (sub-)sequence)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, let I be its input (sub-)sequence and let t range over the accumulation periods starting instants; then I is *persistent* if and only if $card(I \cap [\, t, t + T\,[) > 0, \ \forall t$.

Definition 4.4 (Persistent excitatory/inhibitory input (sub-)sequence)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, let I be its input sub-sequence and let n range over the accumulation periods; then I is *excitatory* (resp.

inhibitory) if and only if $A_n > 0$ (resp. $A_n < 0$) $\forall t$, where A_n is the sum of weighted inputs for the n-th accumulation period.

Definition 4.5 (Persistent constant input (sub-)sequence)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, let I be its input sub-sequence and let n range over the accumulation periods; then I is *constant* if and only if there exists some $K \in \mathbb{Z}$ such that $A_n = K,\ \forall t$.

Definition 4.6 (Periodic output (sub-)sequence)**.** Let $\mathcal{N}$ be a neuron and let $F_{\mathcal{N}} = \{t_1, t_2, \ldots\}$ be its output sequence, then $F_{\mathcal{N}}$ is *periodic* if and only if there exists some $P \in \mathbb{R}^+$ such that $t_{i+1} - t_i = P,\ \forall i$.

Definition 4.7 (Simultaneous input spikes)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, let I be its input sequence, let t range over the accumulation periods starting instants and let $s_1, s_2 \in I$ be two input spikes; then s_1 and s_2 are *simultaneous* if and only if $s_1, s_2 \in [\, t,\, t + T\, [$ for some t.

Definition 4.8 (Consecutive input spikes)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, let I be its input sequence, let t, t' be the starting instants of some accumulation period and the next one, respectively, and let $s_1, s_2 \in I$ be two input spikes; then s_1 and s_2 are *consecutive* if and only if $s_1 \in [\, t,\, t + T\, [\wedge s_2 \in [\, t',\, t' + T\, [$.

Definition 4.9 (Reset times)**.** Let $\mathcal{N}$ be a neuron and let $F_{\mathcal{N}} = \{t_1, t_2, \ldots\}$ be its output sequence, then the set of reset times of $\mathcal{N}$ is $Z_{\mathcal{N}} = \{t + \tau : t \in F_{\mathcal{N}}\}$.

We use calligraphic letters ($\mathcal{A}$) for automata, bold letters ($\mathbf{X}$) for automata states, and lower-case italic letters (t) for automata variables or clocks. Within temporal logic formulae, the predicate $state_{\mathcal{A}}(\mathbf{X})$ is 1 if and only if automaton $\mathcal{A}$ is in state $\mathbf{X}$, 0 otherwise, and $eval_{\mathcal{A}}(t)$ is a function mapping a variable or clock t to the value it currently carries within the context of automaton $\mathcal{A}$: a predicate may consist of the comparison between such a value and a constant. For boolean variables we may abuse the notation writing $eval(b)$ and $\neg eval_{\mathcal{A}}(b)$ instead of $eval_{\mathcal{A}}(b) = 1$ or $eval_{\mathcal{A}}(b) = 0$, respectively.

4.1 Intrinsic Properties

Maximum threshold. Here we show that, assuming an upper bound for the sum of ingoing synapses weights, there exist a way to compute the maximum threshold value such that, any neuron having a threshold greater than or equals to it, will never be able to fire.

Property 4.1 (Threshold-leak factor relation)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron and $a_{max} \in \mathbb{N}^+$ the maximum value of weighted inputs sum, then, if $\theta \geq \frac{a_{max}}{1-\lambda}$, the neuron is not able to fire.

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

$$p_n = a_{max} + \lfloor \lambda \cdot p_{n-1} \rfloor$$

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

$$p_n \leq {p'}_n = a_{max} + \lambda \cdot p'_{n-1}$$

The same inequality can be written in explicit form because of Equation 2.3:

$$p_n \leq {p'}_n = \sum_{k=0}^{n} a_{n-k} \cdot \lambda^k$$

and, since we assumed the neuron always receives a_{max}, a_{n-k} is constant and do not depend on k:

$$p_n \leq a_{max} \cdot \sum_{k=0}^{n} \lambda^k$$

The rightmost factor is a geometric series having a more compact representation:

$$p_n \leq a_{max} \cdot \frac{1 - \lambda^n}{1 - \lambda}$$

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

$$p_n \leq \frac{a_{max}}{1 - \lambda}, \ \forall n \in \mathbb{N}$$

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

Notice that, according to Definition 3.3, synapses weights are never greater than an integer R, so $a_{max} = mR$ for each neuron having m ingoing synapses, even if, in the general case, we will consider $a_{max} = \sum_{i=0}^{m} w_i \leq mR$. We will say that a neuron is *firing enabled* if $\theta < \frac{a_{max}}{1-\lambda}$.

Analysis of neuron timings. We can quantify the amount of time that the neuron requires to complete an accumulate–fire–rest cycle. Such expression is useful to prove some interesting properties, e.g., here we show that there exists a minimum delay between one neuron emission and its successor.

Property 4.2 (Minimum firing period)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a *firing enabled* Synchronous Neuron, then the time difference between successive firings cannot be lower than $T + \tau$.

Proof. Let $A_n = \sum_{k=1}^{T} a_{k+t_0}$ be the sum of weighted inputs during the n-th *accumulation period*, then the neuron behavior can be described as follows:

$$p_n = A_n + \lfloor \lambda \cdot p_{n-1} \rfloor \tag{4.1}$$

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

$$\hat{n} = \underset{n \in \mathbb{N}}{arg\,min} \{p_n : p_n \geq \theta\} \tag{4.2}$$

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

$$\hat{t} = \hat{n} \cdot T + t_0 \tag{4.3}$$

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 t', i.e., the next instant in future when the neuron will complete its refractory period, after having emitted a spike, is:

$$t' = \hat{t} + \tau = \hat{n} \cdot T + \tau + t_0$$

At instant t', the neuron quits its refractory period, n is reset to 0, t_0 is set to t', and $\hat{n}$, $\hat{t}$ and 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 $\hat{n}$:

$$t' - t_0 = \hat{n} \cdot T + \tau \tag{4.4}$$

So, the minimum inter-firing period is $T + \tau$ for $\hat{n} = 1$. Such a property can be verified as follows: let $\mathcal{I}$ be the non-deterministic input generator having $T_{min} = 1$ and, without loss of generality[1], initial delay $D = T + \tau$, then the Timed Automata Network $\mathcal{I} \overset{x}{\parallel} \mathcal{N} \overset{y}{\parallel} \mathcal{O}$ satisfies the following formula:

$$AG(state_{\mathcal{O}}(\mathbf{O}) \implies eval_{\mathcal{O}}(s) \geq T + \tau) \tag{4.5}$$

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 $T + \tau$. □

Analysis of neuron memory. Here we discuss about the neuron capability of taking past events into account when computing its outcome. As argued above, 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.

Definition 4.10 (Neuron inter-emission memory)**.** Let $\mathcal{N}$ be a neuron, let $Z_{\mathcal{N}}$ be its reset times set and let I be an input sub-sequence; then $\mathcal{N}$ has inter-emission *memory* if and only if there exist two different $t, t' \in Z_{\mathcal{N}}$ such that the output sub-sequence produced by $\mathcal{N}$ as a response to I starting from t differs from the output sub-sequence it produces as a response to I starting from t'.

Property 4.3 (Memoryless neuron)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ has not inter-emission memory.

Proof. According to Definition 3.3 each reset time occurs on each $(\mathbf{W} \to \mathbf{A})$ firing. Such event makes $\mathcal{N}$ automaton move back to its initial location while resetting clock t and variables p and a, making them equal to their starting values. So it is impossible for the neuron to behave differently if subjected to the same input sub-sequence. □

[1] the initial delay is required in order to make the formula hold for the first output spike too

Analysis of inhibitory inputs. Here we argue about the effects of an *inhibitory* stimulation to a neuron whose potential lower than its threshold.

Property 4.4 (Inhibitory effect of negative stimulations)**.** Let $\mathcal{N} = (\mathbf{w},\, T,\, \lambda,\, \theta,\, \tau)$ be a Synchronous Neuron, let A_n be the sum of weighted inputs received during the current accumulation period and let p_{n-1} be the neuron potential at the end of the previous accumulation period, then if $p_{n-1} < \theta$ and $A_n < 0$ the neuron cannot fire at the end of the current accumulation period.

Proof. It is sufficient to prove that, under such hypotheses, $p_n < \theta$. Considering p_n definition, we can state that:

$$p_n \leq A_n + \lambda \cdot p_{n-1}$$

so, since A_n is negative, we can rewrite it as $-|A_n|$:

$$p_n + |A_n| \leq \lambda \cdot p_{n-1}$$

and then we deduce:

$$p_n < \lambda \cdot p_{n-1}$$

because $p_n < p_n + |A_n|$ and, consequently:

$$p_n \leq \frac{1}{\lambda} \cdot p_n < p_{n-1}$$

because $\lambda^{-1} \in [\,1,\, \infty\,[$. So finally:

$$p_n < p_{n-1} < \theta$$

□

Next we show that only positive stimulations are necessary for the neuron to produce emissions:

Property 4.5. Let $\mathcal{N} = (\mathbf{w},\, T,\, \lambda,\, \theta,\, \tau)$ be a Synchronous Neuron such that $\theta > 0$, let A_n be the sum of weighted inputs received during the current accumulation period and let p_n be the neuron potential at the end of the current accumulation period, then $p_n \geq \theta \implies A_n > 0$.

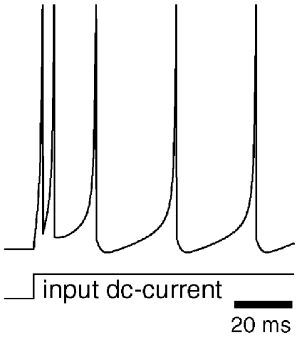


Figure 4.1: Tonic spiking representation for continuous signals from [14].

Proof. It is sufficient to prove that, under such hypotheses, $A_n > 0$. We know that:

$$p_n = A_n + \lfloor \lambda \cdot p_{n-1} \rfloor \geq \theta$$

Let's analyze two sub-cases, with respect to the sign of $\lfloor \lambda \cdot p_{n-1} \rfloor - \theta$:

Consider the case: $\lfloor \lambda \cdot p_{n-1} \rfloor < \theta$. According to the initial hypothesis:

$$\lfloor \lambda \cdot p_{n-1} \rfloor < \theta \leq A_n + \lfloor \lambda \cdot p_{n-1} \rfloor$$

and, consequently:

$$0 < \theta - \lfloor \lambda \cdot p_{n-1} \rfloor \leq A_n$$

so, finally $A_n > 0$.

Consider the case: $\lfloor \lambda \cdot p_{n-1} \rfloor \geq \theta$. This means $p_{n-1} \geq \theta$, in fact:

$$p_{n-1} \geq \lambda \cdot p_{n-1} \geq \lfloor \lambda \cdot p_{n-1} \rfloor \geq \theta$$

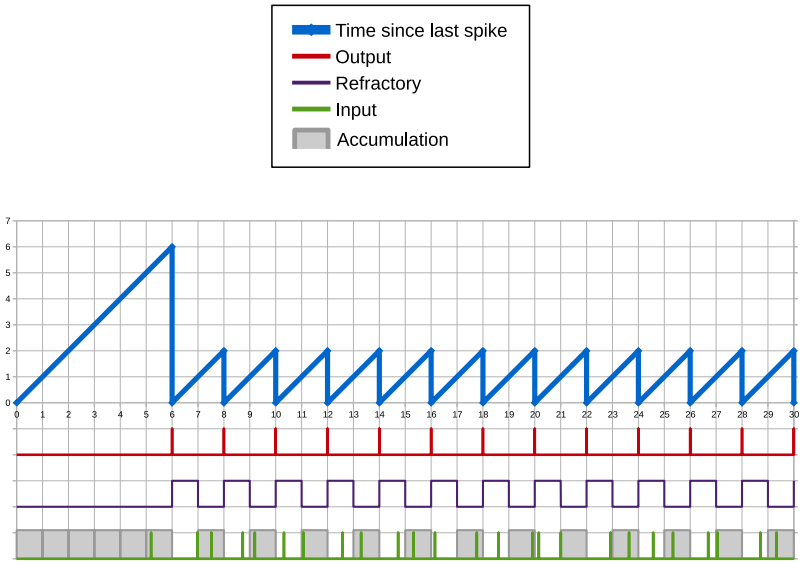
But this is absurd because if $p_{n-1} \geq \theta$, the neuron emits and resets, thus its potential in the next accumulation period is zero, which is in contradiction with the hypothesis $p_n \geq \theta > 0$.

□

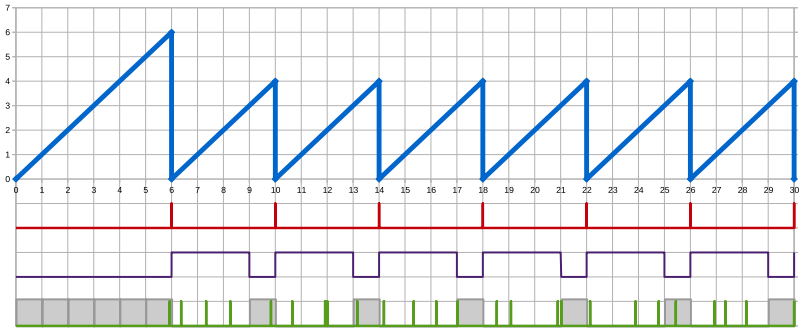
4.2 Capabilities

Tonic Spiking. "Tonic spiking" is the behavior of a neuron producing a periodic output sub-sequence as a response to a persistent excitatory constant input sub-sequence. An example is shown in Figure 4.1.

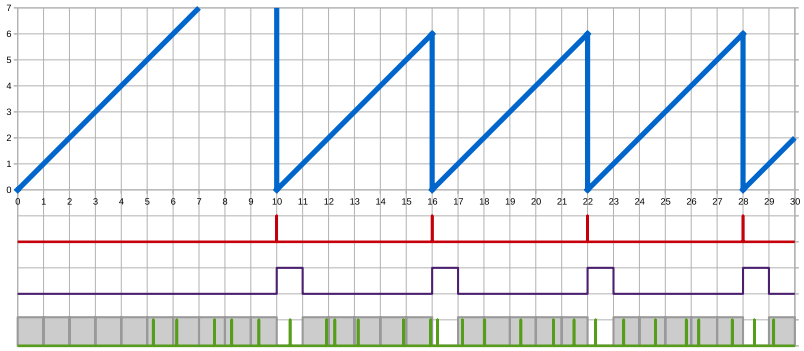
Property 4.6 (Tonic spiking)**.** Let $\mathcal{N} = (w, T, \lambda, \theta, \tau)$ be a Synchronous Neuron having only one ingoing excitatory synapse such that $w > 0$ and



(a) $R = 1000$, $m = 1$, $w = 1000$, $T = 1$, $\lambda = \frac{1}{3}$, $\theta = 100$, $\tau = 1$



(b) $R = 1000$, $m = 1$, $w = 1000$, $T = 1$, $\lambda = \frac{9}{10}$, $\theta = 900$, $\tau = 3$



(c) $R = 1000$, $m = 1$, $w = 1000$, $T = 1$, $\lambda = \frac{1}{2}$, $\theta = 1900$, $\tau = 1$

Figure 4.2: Tonic spiking simulations. Each diagram shows the time elapsed since last neuron emission (blue), the emitted spikes (red), the refractory periods (purple) and input spikes (green) within accumulation periods (gray) for three different neurons and for 30 time units. In each case, the input generator is a fixed-rate generator having initial delay 5 and time window size 1.

$\theta < w/(1-\lambda)$ and let $\mathcal{I}$ be the input source connected to $\mathcal{N}$ producing a persistent input sequence, then $\mathcal{N}$ produces a periodic output sequence.

Proof (Sketch). Let $\mathcal{I}$ be the fixed-rate input generator having arbitrary initial delay D and time window size T, and let $\mathcal{O}$ be an output consumer, then the Timed Automata Network $\mathcal{I} \overset{x}{\parallel} \mathcal{N} \overset{y}{\parallel} \mathcal{O}$ satisfies the following formulae:

$$\begin{cases} state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{O}}(e) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O}) \wedge \neg eval_{\mathcal{O}}(e) \\ state_{\mathcal{O}}(\mathbf{O}) \wedge \neg eval_{\mathcal{O}}(e) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{O}}(e) \end{cases} \tag{4.6}$$

where $\mathbf{O}$ is the location that automaton $\mathcal{O}$ reaches after consuming a spike and e is boolean variable whose value changes whenever $\mathcal{O}$ moves into location $\mathbf{O}$. So, whenever automaton $\mathcal{O}$ reaches location $\mathbf{O}$ it will eventually reach it again. As shown in Figure 4.2, if we simulate neurons having different parameters providing them the same input $\mathcal{I}$, then they keep producing a periodic outcome whose period only depends on T and τ as long as $\theta < \frac{w}{1-\lambda}$. □

It should be noted that one may also find the value P of the period of some given neuron $\mathcal{N}$ by means of simulations, thus the periodic behavior can be proven by a model-checker verifying the following formula:

$$AG(state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{N}}(f) \implies eval_{\mathcal{O}}(s) = P) \tag{4.7}$$

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

Integrator. "Integrator" is the behavior of a neuron producing an output spike whenever it receives *at least* a specific number of simultaneous spikes from different input sources or when it receives a certain amount of *consecutive* spikes from a specific input source. So the neuron parameters can be tuned in order to *detect* (i.e., fire as a consequence of) a given number of simultaneous or consecutive spikes. An example is shown in Figure 4.3.

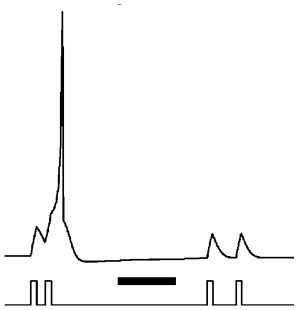


Figure 4.3: Integrator behavior representation for continuous signals, from [14].

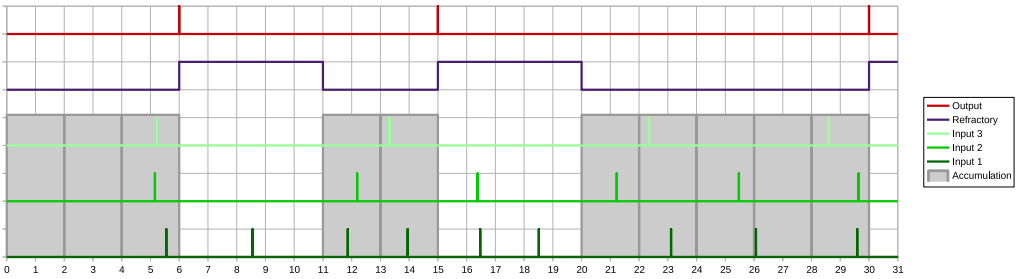
Property 4.7 (Simultaneous integrator)**.** Let $\mathcal{N} = ((R, \ldots, R), T, \lambda, n, \tau)$ be a Synchronous Neuron having m synapses with maximum excitatory weight R and an integer threshold $n \leq m$, then the neuron emits if it receives a spike from at least n input sources during the same accumulation period.

Proof (Sketch). Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be non-deterministic input generators constrained to wait more than T time units between an emission and its successor, and let $\mathcal{O}$ be an output consumer, then the Timed Automata Network $(\mathcal{I}_1, \ldots, \mathcal{I}_m) \overset{\mathbf{x}}{\parallel} \mathcal{N} \overset{y}{\parallel} \mathcal{O}$ satisfies the following formula stating that, if at least n generators are in location $\mathbf{S}$ while $\mathcal{N}$ is in $\mathbf{A}$, then $\mathcal{O}$ will eventually capture an output of $\mathcal{N}$:

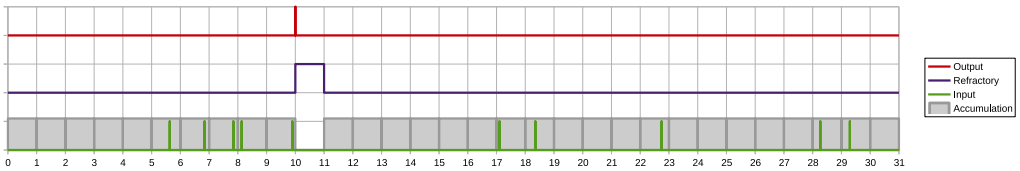
$$\left(\sum_{i=1}^{m} state_i(\mathbf{S}) \geq n \right) \wedge state_{\mathcal{N}}(\mathbf{A}) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O}) \tag{4.8}$$

where $\mathbf{S}$ is the location that each automaton $\mathcal{I}_i$ reaches after producing a spike and $\mathbf{A}$ is the accumulation location of the neuron $\mathcal{N}$. As shown in Figure 4.4a, a neuron, under such hypotheses, will fire as soon as it receive n simultaneous spikes. □

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.



(a) $R = 1000$, $m = 3$, $w_1, w_2, w_3 = 1000$, $T = 2$, $\lambda = \frac{1}{2}$, $\theta = 3000$, $\tau = 5$



(b) $R = 1000$, $m = 1$, $w = 1000$, $T = 1$, $\lambda = \frac{1}{2}$, $\theta = 1900$, $\tau = 1$

Figure 4.4: Chart 4.4a represents the behavior of a neuron, having 3 ingoing synapses, which is able to detect the simultaneity of at least 3 inputs: whenever two or more input spikes (green lines) occur during the same accumulation period (gray), an output spike is produced (red). Chart 4.4b represents the behavior of a neuron, having a single ingoing synapse, which is able to detect a sequence of 5 consecutive input spikes.

Property 4.8 (Sequential integrator)**.** Let $\mathcal{N} = (w, T, \lambda, \theta, \tau)$, be a Synchronous Neuron having only one ingoing synapse, such that $\theta < \frac{w}{1-\lambda}$, then there exists a maximal sequence of consecutive input spikes of length $\hat{n}$ that results in an output spike.

Proof (Sketch). Let $\mathcal{I}$ be the fixed-rate input generator having arbitrary initial delay D and time window size T, let $\mathcal{O}$ be an output consumer, and let $\hat{n}$ be the minimum amount of consecutive input spikes required to make the potential overcome the threshold, obtained by means of simulation or by recursively computing p_n until it reaches the threshold value; then the Timed Automata Network $\mathcal{I} \overset{x}{\parallel} \mathcal{N} \overset{y}{\parallel} \mathcal{O}$ satisfies the following formula stating that, whenever $\mathcal{O}$ receives a spike, the number of consecutive spikes never greater than $\hat{n}$:

$$AG(state_{\mathcal{O}}(\mathbf{O}) \implies eval_{\mathcal{N}}(c) \leq \hat{n}) \tag{4.9}$$

where c is an integer variable of automaton $\mathcal{N}$ counting the amount of consecutive accumulation periods that received at least one spike since last emission. □

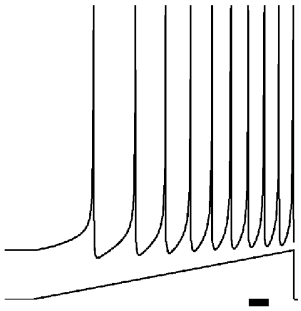


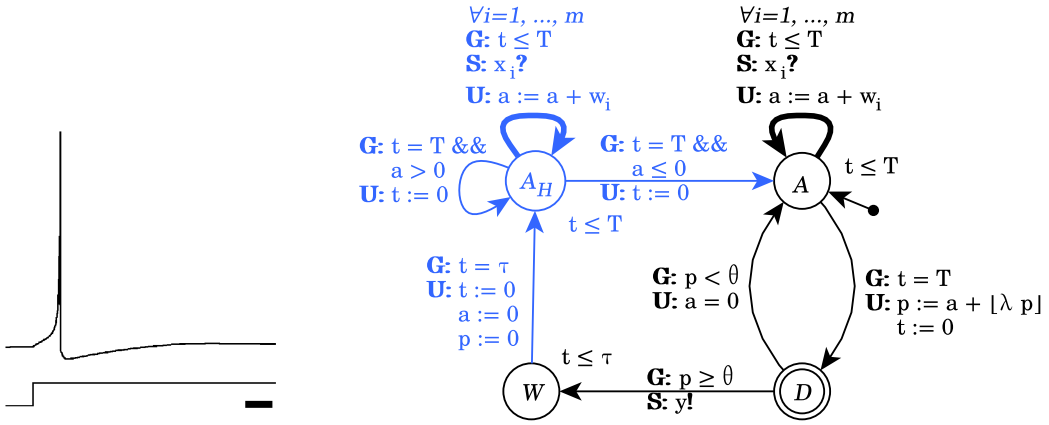
Figure 4.5: Excitability capability representation for continuous signals, from [14].

Figure 4.4b shows a simulation of a neuron able to detect 5 consecutive spikes.

Excitability. "Excitability" is the behavior of a neuron emitting sequences having a *decreasing* inter-firing period, i.e., and increasing output frequency, when stimulated by an *increasing* number of excitatory inputs. An example is shown in Figure 4.5

Property 4.9 (Excitability)**.** Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a *firing enabled* Synchronous 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, e.g., an increasing number of input sources emitting persistent inputs, then a_t is the non-negative, non-decreasing (i.e., $a_{t+1} \geq a_t,\ \forall t$) 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, $A_n = \sum_{k=1}^{T} a_{k+t_0}$ 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, according to Equation 4.1 and Equation 4.2, the following statement holds: if A_n increases then $\hat{n}$ decreases. Being $\hat{n}$ the only variable in Equation 4.4, if $\hat{n}$ decreases then the difference $t' - t_0$ decreases too. □



(a) Phasic spiking representation for continuous signals from [14].

(b) The Synchronous neuron model edited to be able to reproduce the phasic spiking behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored.

Figure 4.6: Phasic Spiking: example and model variant.

4.3 Limits

Phasic Spiking. "Phasic spiking" is the behavior of a neuron producing a *single* output spike on the onset of a persistent and excitatory input subsequence and then remaining quiescent until the end of such sub-sequence. An example is shown in Figure 4.6a.

Such a behavior requires the neuron to be able to detect the onset of an excitatory input sub-sequence and, therefore, it depends on the neuron to have inter-emission memory.

Property 4.10. Let $\mathcal{N}$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Phasic Spiking* behavior.

Proof. It is sufficient to prove that such a behavior requires the neuron to have inter-emission memory. In fact, the phasic spiking behavior 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 sub-sequence, which is impossible for a memoryless neuron, as stated in Property 4.3. □

The synchronous model can be edited as shown in Figure 4.6b in order

to make it able to reproduce such a behavior. This variant simply makes the neuron able to "remember" if it is receiving a persistent excitatory input sub-sequence. After each refractory period, the neuron moves to location $\mathbf{A_H}$, instead of moving back to $\mathbf{A}$. Here, accumulation periods keep repeating every T time units and weighted inputs are accumulated in variable a, as for location $\mathbf{A}$. The difference between $\mathbf{A_H}$ and $\mathbf{A}$ is that the former one *ignores* positive values of a at the end of each accumulation period. Conversely, a non-positive value of a, at the end of some accumulation period, leads the neuron back in location $\mathbf{A}$. So, such a variant of the Synchronous Model will fire only one spike on the onset of each persistent excitatory input sub-sequence.

Tonic Bursting. A *burst* is a finite sequence of *high frequency* spikes. Some behaviors presented in [14], e.g., tonic of phasing bursting, require the neuron to be able to generate output bursts instead of single spikes. Here we formalize the "burst" concept and discuss about the tonic bursting behavior.

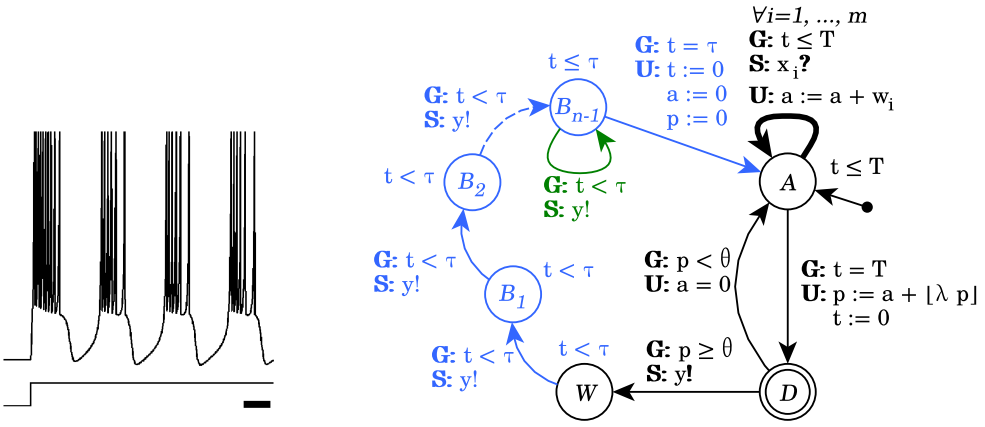
Definition 4.11 (Burst)**.** A spike sub-sequence is a *Burst* if it is composed by a least a given number of spikes having an occurrence rate greater than $1/\tau$, where τ is the refractory period duration of the neuron generating the sub-sequence.

Definition 4.12 (Burst sequence)**.** A burst sequence is a spike sequence composed by bursts, subjected to the following constraint: the time difference between the last spike of each burst and the first spike of the next burst it greater than τ, where τ is the refractory period duration of the neuron generating the sequence.

Thus, "Tonic Bursting" is the behavior of a neuron producing a burst sub-sequence as a response to a persistent and excitatory input sub-sequence. An example is shown in Figure 4.7a.

Such a behavior, differently from Phasic Spiking, does not require the neuron to have inter-emission memory but it requires the neuron to be able to produce bursts.

Property 4.11. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot produce bursts.



(a) Tonic bursting representation for continuous signals from [14].

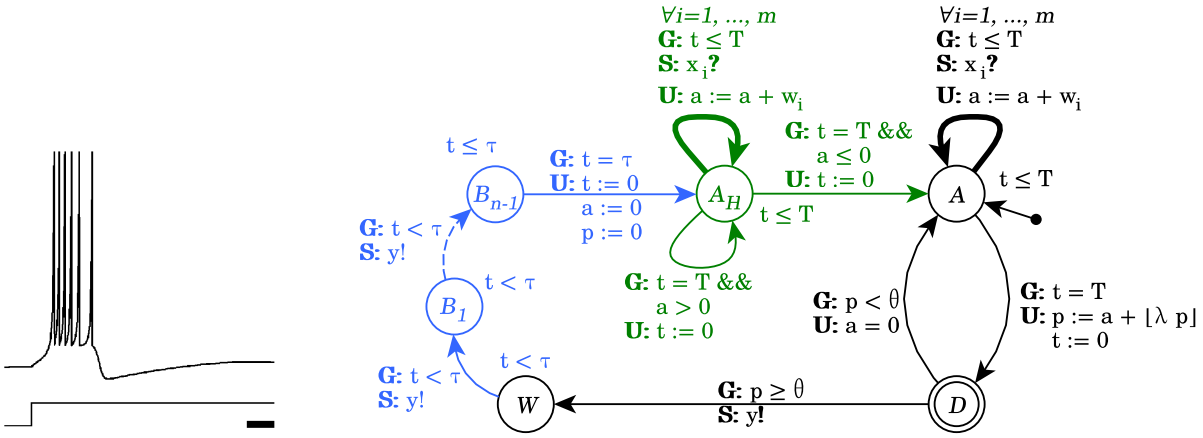
(b) The Synchronous neuron model edited to be able to reproduce the Tonic Bursting behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored. Let n be the number of spikes a burst is composed of, then such automaton has $n-1$ locations $\mathbf{B}_i$. One may include the green-colored part to model bursts composed by *at least* n spikes.

Figure 4.7: Tonic Bursting: example and model variant.

Proof. $\mathcal{N}$ cannot emit spikes having a rate greater than $1/(T+\tau)$, as stated by Property 4.2, so it cannot produce bursts. □

Corollary. $\mathcal{N}$ cannot reproduce the *Tonic Bursting* behavior.

The Synchronous Model can be edited as shown in Figure 4.7b in order to make it able to reproduce such a behavior. This variant simply makes the neuron emit bursts instead of spikes. The Synchronous Model is redefined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau, n)$ where n is the number of spikes that each burst emitted by $\mathcal{N}$ will contain. If we also consider the green $(\mathbf{B}_{n-1} \to \mathbf{B}_{n-1})$ loop of Figure 4.7b, then n is the *minimum* number of spikes composing a burst. The rationale of this edit is straightforward: at the end of each refractory period, the neuron must iterate over $n-1$ locations $\mathbf{B}_i$ until eventually reaching location $\mathbf{A}$, as usual. On each ingoing edge of each location $\mathbf{B}_i$, a spike is fired. Because of the invariants of locations $\mathbf{B}_i$, the entire burst emission cannot last longer than the refractory period. Please note that, if $n = 1$, then $\mathbf{B}_{n-1} \equiv \mathbf{W}$ and the automaton degenerates to the original Synchronous Neuron structure of Figure 3.2.



(a) Phasic bursting representation for continuous signals from [14].

(b) The Synchronous neuron model edited to be able to reproduce the Phasic Bursting behavior. Variations with respect to the original model shown in Figure 3.2 are colored. Note that this variant is achieved by mixing the Phasic Spiking (green part) and Tonic Bursting (blue part) variants.

Figure 4.8: Phasic Bursting: example and model variant.

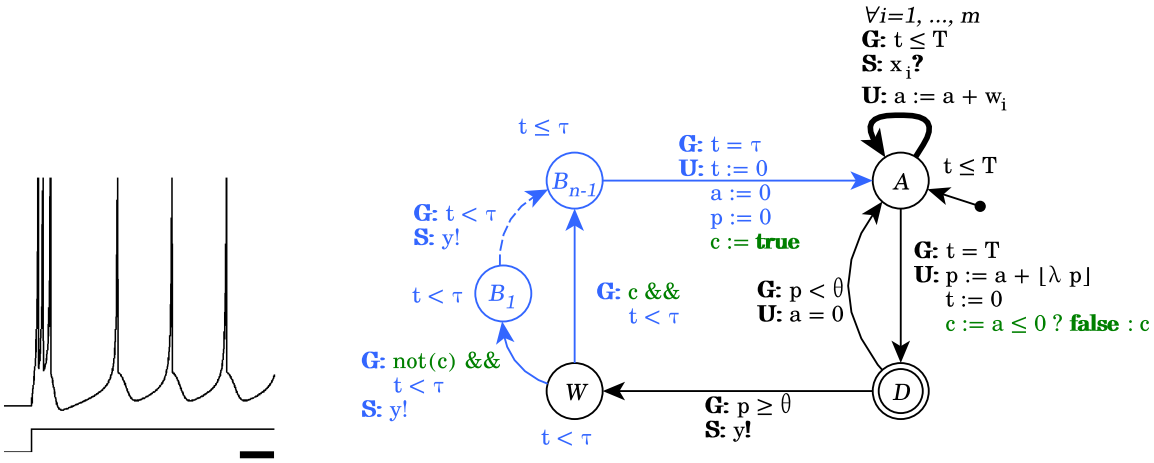
Phasic Bursting. "Phasic Bursting" is the behavior of a neuron producing a burst *on the onset* of a persistent excitatory input sub-sequence and then remaining quiescent until the end of such sub-sequence. An example is shown in Figure 4.8a.

Such a behavior, similarly to Phasic Spiking, requires the neuron to have inter-emission memory, in order to detect the beginning of an excitatory input sub-sequence, and, analogously to Tonic Bursting, depends on the neuron to be able to produce bursts.

Property 4.12. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Phasic Bursting* behavior.

Proof. According to Property 4.11, $\mathcal{N}$ cannot produce bursts, thus it cannot reproduce the *Phasic Bursting* behavior. □

The Synchronous Model can be edited as shown in Figure 4.8b in order to make it able to reproduce such a behavior. This variant simply merges the edits proposed for Phasic Spiking (green part) and Tonic Bursting (blue part).



(a) Bursting-then-Spiking representation for continuous signals from [14].

(b) The Synchronous neuron model edited to be able to reproduce the Bursting-then-Spiking behavior. Variations with respect to the original model shown in Figure 3.2 are colored. The blue-colored part is needed to produce bursts while the green-colored edits are needed to keep track of the onset of an excitatory input sub-sequence.

Figure 4.9: Bursting-then-Spiking: example and model variant.

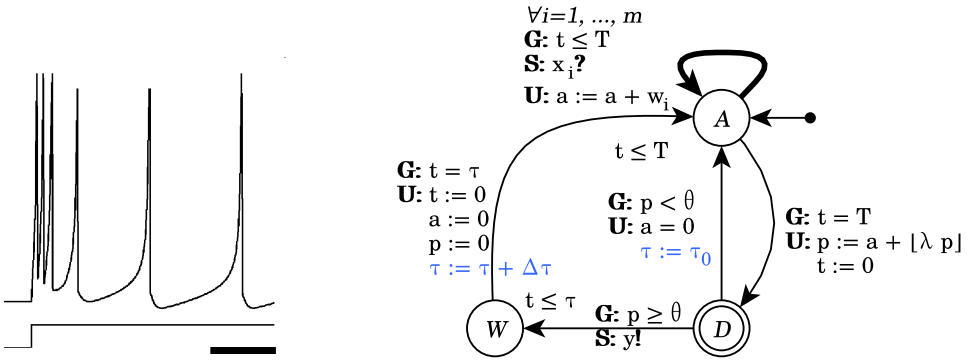
Bursting-then-Spiking. "Bursting-then-Spiking" is the behavior of a neuron producing a burst *on the onset* of a persistent excitatory input subsequence and then producing a periodic output sub-sequence until the end of such sub-sequence. An example is shown in Figure 4.9a.

Such a behavior, similarly to Phasic Bursting, requires the neuron to have inter-emission memory, in order to detect when a persistent subsequence is beginning, and depends on the neuron to be able to produce bursts.

Property 4.13. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Bursting-then-Spiking* behavior.

Proof. According to Property 4.11, $\mathcal{N}$ cannot produce bursts, thus it cannot reproduce the *Bursting-then-Spiking* behavior. □

The Synchronous Model can be edited as shown in Figure 4.9b in order to make it able to reproduce such a behavior. This variant, similarly to the one proposed for Tonic Bursting, comprehends locations $\mathbf{B}_1, \ldots, \mathbf{B}_{n-1}$, allowing it to produce n-bursts. Moreover, the model is extended by means of the c



(a) Spike Frequency Adaptation representation for continuous signals from [14].

(b) The Synchronous neuron model edited to be able to reproduce the Spike Frequency Adaptation behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored.

Figure 4.10: Spike Frequency Adaptation: example and model variant.

boolean variable keeping track of persistent excitatory input sub-sequences. More precisely, c is set at the end of each *refractory* period and it is reset at the end of any *accumulation* period where the sum of weighted input is non-positive. So, at the end of an accumulation period, if the neuron just emitted one spike and $c = false$ (i.e., it's the first output spike since the beginning of the current input sub-sequence), it will emit $n - 1$ more spikes in order to compose a burst. Conversely, if $c = true$, then it will not produce any further spike until the end of the next accumulation period.

Spike Frequency Adaptation. "Spike Frequency Adaptation" is the behavior of a neuron producing a decreasing-frequency output sub-sequence as a response to a persistent excitatory input sub-sequence. The inter-emission time difference increases as the time elapsed since the onset of the input sub-sequence and resets to the initial value at the end of such a sub-sequence. An example is shown in Figure 4.10a.

This behavior requires the neuron to have inter-emission memory: it should be able to keep track of the time elapsed since the beginning of the input sub-sequence.

Property 4.14. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Spike Frequency Adaptation* behavior.

Proof. It is sufficient to prove that such behavior requires the neuron to have inter-emission memory. In fact, the Spike Frequency Adaptation behavior requires the neuron to detect the beginning instant of an excitatory input sub-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 for any Synchronous Neuron, as stated in Property 4.3. □

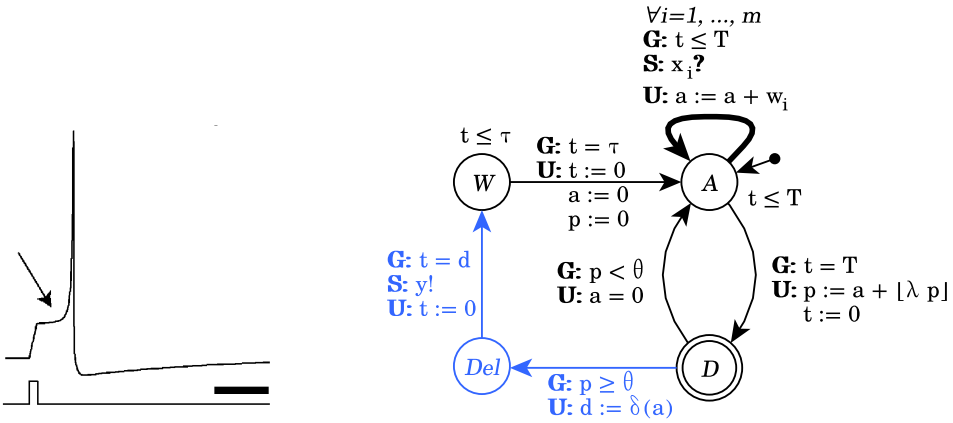
The Synchronous Model can be edited as shown in Figure 4.10b in order to make it able to reproduce such a behavior. This variant allows the refractory period to increase after each neuron emission thus making the output frequency decrease. More precisely, the Synchronous Model is re-defined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau_0, \Delta\tau)$ where $\tau_0 \in \mathbb{N}^+$ is the default refractory period duration and $\Delta\tau \in \mathbb{N}^+$ represents the refractory period variation, while τ is a variable of automaton $\mathcal{N}$. The initial value of variable τ is τ_0. On every firing of edge $(\mathbf{W} \to \mathbf{A})$ the variable is increased of $\Delta\tau$, while, on every firing of $(\mathbf{D} \to \mathbf{A})$, it is reset to τ_0. So, any persistent excitatory input sub-sequence will lead to an output sub-sequence having a decreasing frequency.

Spike Latency. "Spike Latency" is the behavior 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 lead it to emission, i.e., for a Synchronous Neuron, the sum of weighed inputs received during the accumulation period preceding the emission. An example is shown in Figure 4.11a.

This behavior does not require the neuron to have inter-emission memory, nevertheless it requires the neuron to be able to postpone its outcome.

Property 4.15. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Spike Latency* behavior.

Proof. Let n be the first accumulation period since the last reset time where the potential reaches or overcomes θ. Then, according to Definition 3.3, the neuron emission instant will be exactly $n \cdot T$: because of location $\mathbf{D}$ being *committed*, there is no way for the neuron to postpone its firing instant. □



(a) Spike Latency representation for continuous signals from [14].

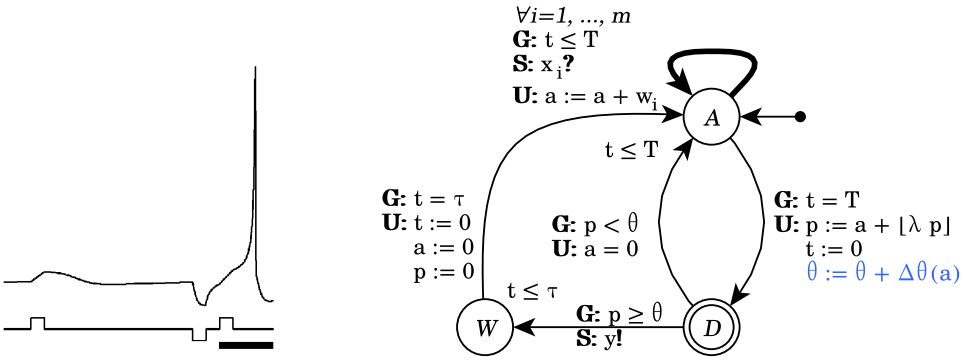
(b) The Synchronous neuron model edited to be able to reproduce the Spike Latency behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored.

Figure 4.11: Spike Latency: example and model variant.

The Synchronous Model can be edited as shown in Figure 4.11b in order to make it able to reproduce such a behavior. The proposed variant introduces a delay between the instant the neuron reaches or overcomes its threshold and actual emission instant. Such a delay depends solely on the sum of weighted inputs from the last accumulation period. More formally, the Synchronous Model is re-defined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau, \delta)$ where $\delta : \mathbb{N} \to \mathbb{N}$ is the function computing the delay according to the sum of weighted inputs. At the end of each accumulation period, if the potential is greater than or equal to the threshold, the neuron will compute the delay duration $\delta(a)$, assigning it to an integer variable d and then wait in location **Del** for d time units before emitting a spike on channel y.

Threshold Variability. "Threshold variability" is the behavior of a neuron allowing its threshold to vary according to the strength of its income. More precisely, an excitatory input will rise the threshold while an inhibitory input will decrease it. As a consequence of such a behavior, excitatory inputs may more easily lead the neuron to fire when occurring after an inhibitory input, as shown in the example of Figure 4.12a.

This behavior does not require the neuron to have inter-emission memory,



(a) Threshold Variability representation for continuous signals from [14]: the inhibitory spike decreases the neuron threshold making the following excitatory spike sufficient to make the neuron fire.

(b) The Synchronous neuron model edited to be able to reproduce the Threshold variability behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored.

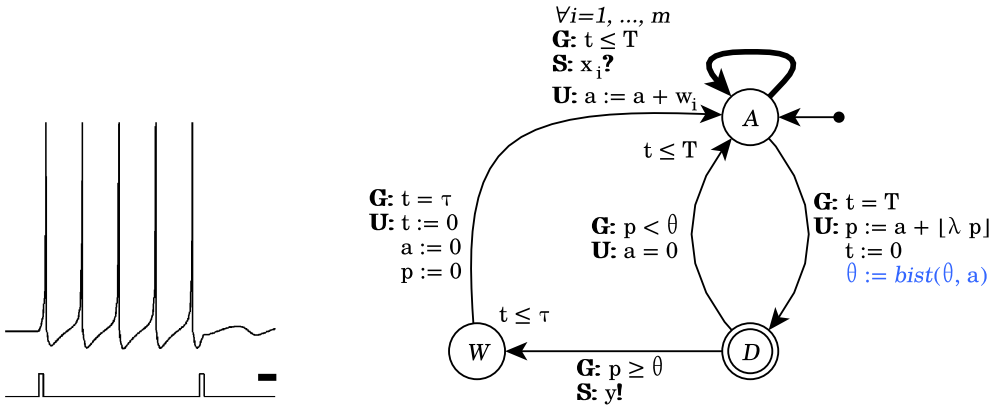
Figure 4.12: Threshold variability: example and model variant.

nevertheless it requires the neuron threshold to vary according to its inputs.

Property 4.16. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Threshold Variability* behavior.

Proof. This is true by construction according to Definition 3.3: the neuron threshold never changes. □

The Synchronous Model can be edited as shown in Figure 4.12b in order to make it able to reproduce such a behavior. This variant allows the threshold to vary after each accumulation period according to the current sum of weighted inputs. More precisely, the Synchronous Model is re-defined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta_0, \tau, \Delta\theta)$ where $\theta_0 \in \mathbb{N}^+$ is the initial threshold and $\Delta\theta : \mathbb{Z} \to \mathbb{Z}$ represents the threshold variation function, while θ is a variable of automaton $\mathcal{N}$. The threshold variable initial value is θ_0. On every firing of edge $(\mathbf{A} \to \mathbf{D})$ the threshold variable is increased of $\Delta\theta(a)$, which is an integer value whose sign is opposite to the sign of a and whose magnitude is proportional to the magnitude of a, where a is the sum of weighted inputs occurred during the last accumulation period.



(a) Bistability representation for continuous signals from [14].

(b) The Synchronous neuron model edited to be able to reproduce the Bistability behavior. Variations with respect to the original model shown in Figure 3.2 are blue-colored.

Figure 4.13: Bistability: example and model variant.

Bistability. "Bistability" is the behavior of a neuron alternating between two modes of operation: *periodic emission* and *quiescence*. During the former mode, it emits a periodic output sub-sequence, even if it receives no excitatory spike. During the quiescent mode, it does not emit. The neuron switches from one mode to the other every time it receives an excitatory spike. An example is shown in Figure 4.13a.

Such a behavior requires the neuron to (i) be able to produce a periodic output sub-sequence, even if no excitatory spike is received, (ii) be able to *not* produce any output when no spike is received, (iii) be able to switch between the two modes of operation when an excitatory spike is received.

Property 4.17. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Bistability* behavior.

Proof. It is sufficient to prove that (i) $\mathcal{N}$ cannot produce a periodic output sub-sequence if no excitatory spike is received, or (ii) $\mathcal{N}$ cannot remain quiescent if no spike is received, or (iii) $\mathcal{N}$ cannot switch between the two modes of operation. If $\theta = 0$ and no excitatory spike is received, $\mathcal{N}$ produces a periodic output sub-sequence: it is a degenerate case of Property 4.6. Conversely, if $\theta > 0$ and no input is received, then $\mathcal{N}$ remains quiescent. Since,

by construction, the threshold cannot vary, there is no way for the neuron to switch between the two modes of operation. □

The Synchronous Model can be modified as shown in Figure 4.13b in order to make it able to reproduce such a behavior. This variant simply 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 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. More precisely, the Synchronous Model is re-defined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta_0, \tau)$ where $\theta_0 \in \mathbb{N}^+$ is the initial threshold value, while θ is a variable of automaton $\mathcal{N}$. On every firing of edge $(\mathbf{A} \rightarrow \mathbf{D})$, i.e., at the end of every accumulation period, the threshold value θ is computed by means of a function $bist(\cdot)$ defined as follows:

$$bist(\theta, a) = \begin{cases} 0 & if\ \theta > 0 \wedge a > 0 \\ \theta_0 & if\ \theta = 0 \wedge a > 0 \\ \theta & if\ a \leq 0 \end{cases}$$

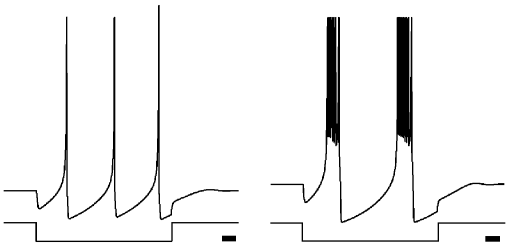
So every accumulation period where the sum of weighted inputs is positive makes the neuron threshold switch between the 0 and θ_0 values.

Inhibition-induced activities. "Inhibition-induced Spiking" (resp. "Bursting") is the behavior of neuron producing a spike (resp. burst) output subsequence as a response to a persistent *inhibitory* input sub-sequence. Examples are shown in Figure 4.14.

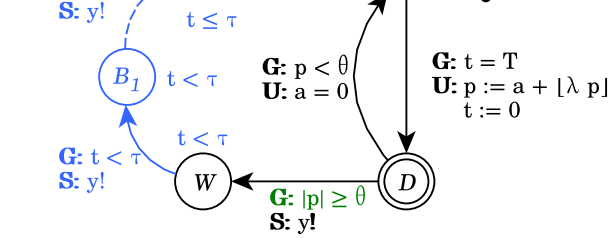
Both behaviors requires the neuron to be able to emit as a consequence of some inhibitory input spikes. Particularly, Inhibition induced *Bursting* also depends on the neuron to be able to produce bursts.

Property 4.18. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Inhibition-induced Spiking* or *Inhibition-induced Bursting* behavior.

Proof. It is sufficient to recall that inhibitory input spikes cannot lead $\mathcal{N}$ to emit according to Property 4.5. Moreover, *Inhibition-induced Bursting*



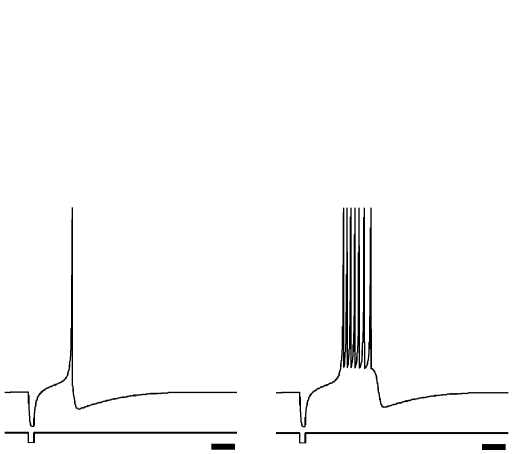
(a) Inhibition-induced spiking.



(b) Inhibition-induced bursting.

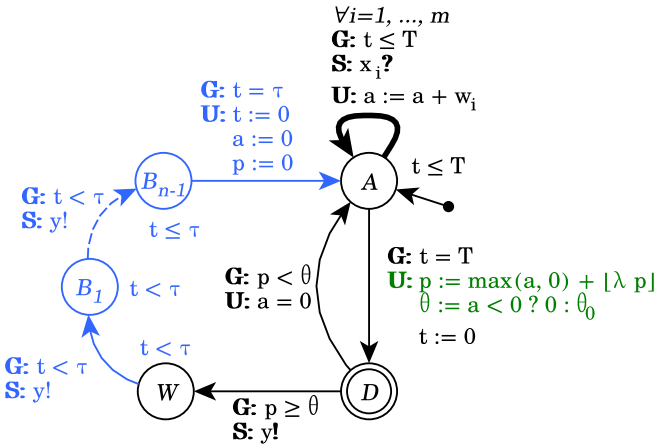
(c) The Synchronous neuron model edited to be able to produce Inhibition-induced activities. Variations with respect to the original model shown in Figure 3.2 are blue-colored. The blue part allows the neuron to emit bursts. The degenerate case $n = 1$ simply emits spikes. The green edit allows the neuron to after a number of inhibitory inputs.

Figure 4.14: Inhibition-induced activities: examples and model variants. Images, representing continuous signals, are from [14].



(a) Rebound spike.

(b) Rebound burst.



(c) The Synchronous neuron model edited to be able to produce Inhibition-induced activities. Variations with respect to the original model shown in Figure 3.2 are blue-colored. The blue part allows the neuron to emit bursts. The degenerate case $n = 1$ simply emits spikes. The green edit allows the neuron to produce a rebound spike/burst after an inhibitory input.

Figure 4.15: Rebound activities: examples and model variants. Images, representing continuous signals, are from [14].

cannot be reproduced by $\mathcal{N}$ because the latter cannot produce bursts, as stated by Property 4.11. □

The Synchronous Model can be edited as shown in Figure 4.14c in order to make it able to reproduce such behaviors. For what concerns the Inhibition-induced Spiking behavior, we propose a variant where the neuron emits whenever *the absolute value* of its potential reaches or overcomes the threshold. The Inhibition-induced Bursting behavior is obtained by adding locations $\mathbf{B}_1, \ldots, \mathbf{B}_{n-1}$ as described in the Tonic Bursting paragraph.

Rebound activities. "Rebound Spike" (resp. "Burst") is the behavior of a neuron producing an output spike (resp. burst) after it received an inhibitory input. Examples are shown in Figure 4.15.

Similarly to Inhibition-induced activities, these behaviors require the neuron to be able to emit as a consequence of an inhibitory input spike. Further-

more, Rebound *Bursting* also depends on the neuron to be able to produce bursts.

Property 4.19. Let $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta, \tau)$ be a Synchronous Neuron, then $\mathcal{N}$ cannot reproduce the *Rebound Spiking* or *Rebound Bursting* behavior.

Proof. It is sufficient to recall that inhibitory input spikes cannot lead $\mathcal{N}$ to emit according to Property 4.5. Moreover, Rebound *Bursting* cannot be reproduced by $\mathcal{N}$ because the latter cannot produce bursts, as stated by Property 4.11. □

The Synchronous Model can be edited as shown in Figure 4.15c in order to make it able to reproduce such behaviors. For what concerns the Rebound Spiking behavior, we propose a variant where the neuron potential is always non-negative and the threshold is set to 0 by inhibitory stimulations. We recall that a null threshold would make the neuron emit even if its potential is 0. More precisely, the Synchronous Model is re-defined as a tuple $\mathcal{N} = (\mathbf{w}, T, \lambda, \theta_0, \tau)$ where $\theta_0 \in \mathbb{N}^+$ is the nominal threshold value, while θ is a variable of automaton $\mathcal{N}$. On every firing of the edge $(\mathbf{A} \to \mathbf{D})$, i.e., at the end of every accumulation period, if the current sum of weighted inputs a is negative, the threshold θ is set to 0, otherwise it is set to θ_0. Thus, an inhibitory stimulus will produce a rebound spike. The Rebound *Bursting* behavior is obtained by adding locations $\mathbf{B}_1, \ldots, \mathbf{B}_{n-1}$ as described in the Tonic Bursting paragraph.

Chapter 5

Conclusions and future works

In this report we formalized the LI&F model of Spiking Neural Networks via Timed Automata Networks. LI&F neurons are modeled as automata waiting for inputs on a number of different channels, for a fixed amount of time. When such *accumulation period* is over, the current *potential* value is computed by means of a recursive formula 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 formalized by a set of automata one for each neuron, running in parallel and sharing channel 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 modeling unpredictable sequences.

We validated our neuron model proving some characteristic properties expressed in CTL via model-checking:

- it is able to exhibit the *tonic spiking* behavior, i.e., it periodically emits a spike if stimulated by a persistent excitatory input;
- it is able to act as an *integrator* under some proper parameter settings, i.e., it can detect — meaning that it fires as a consequence of — a

defined amount of simultaneous or consecutive input spikes;

- it is *excitable*: its output frequency increases (i.e., its inter-emission period decreases) if its total stimulus magnitude keeps increasing;
- there exists a way to compute the *maximum threshold*, i.e., the threshold value such that any greater or equal value would prevent the neuron from firing.

Future research directions. We consider this work as the starting point for a number of research directions: we plan to study whether our model complies to the definition of LI&F and thus cannot reproduce behaviors requiring *bursts* emission capability, as stated in [14] (e.g., tonic or phasic bursting), or some notion of *memory* (e.g., phasic spiking, or bistability). Furthermore, it may be interesting to produce analogous formalizations for more complex spiking neuron models like, e.g., the *theta-neuron* model [8] or Izhikevich's one [13]. In a wider perspective we would also like to model networks composed by more than one neuron taking as starting point our formalization into Timed Automata like e.g., *mutual inibition networks* [18]. 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 [10].

Bibliography

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. Connectionist models and their implications: Readings from cognitive science. chapter A Learning Algorithm for Boltzmann Machines, pages 285–307. Ablex Publishing Corp., Norwood, NJ, USA, 1988.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[3] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal 4.0.* Department of Computer Science, Aalborg University, Denmark, 11 2006.

[4] Johan Bengtsson and Wang Yi. *Timed Automata: Semantics, Algorithms and Tools*, pages 87–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[5] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT Press, Cambridge, MA, USA, 1999.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[7] Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. In *Fifth International Workshop on Hybrid Systems Biology*, Grenoble, France, October 2016.

[8] G. B. Ermentrout and N. Kopell. Parabolic bursting in an excitable system coupled with a slow oscillation. *SIAM Journal on Applied Mathematics*, 46(2):233–253, 1986.

[9] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[10] Donald O. Hebb. *The Organization of Behavior*. John Wiley, 1949.

[11] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[12] J. J. Hopfield. Neurocomputing: Foundations of research. chapter Neural Networks and Physical Systems with Emergent Collective Computational Abilities, pages 457–464. MIT Press, Cambridge, MA, USA, 1988.

[13] E. M. Izhikevich. Simple model of spiking neurons. *Trans. Neur. Netw.*, 14(6):1569–1572, November 2003.

[14] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept 2004.

[15] L. Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635, 1907.

[16] Wolfgang Maass. On the Computational Complexity of Networks of Spiking Neurons. Technical report, Institute for Theoretical Computer Science, Technische Universitaet Graz, 05 1994.

[17] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

[18] Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, 56(5):345–353, 1987.

[19] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[20] Hélène Paugam-Moisy and Sander Bohte. *Computing with Spiking Neuron Networks*, pages 335–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] Michael Recce. Pulsed neural networks. chapter Encoding Information in Neuronal Activity, pages 111–131. MIT Press, Cambridge, MA, USA, 1999.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.