Modelling and Formal Verification of Neuronal Archetypes Coupling

Elisabetta de Maria, Thibaud L 'Yvonnet, Daniel Gaffé, Annie Ressouche, Franck Grammont

Modelling and Formal Verification of Neuronal Archetypes Coupling

Elisabetta De Maria[1], Thibaud L'Yvonnet[1], Daniel Gaffé[2], Annie Ressouche[3], and Franck Grammont[4]

[1] *Université Côte d'Azur, CNRS, I3S, France*
[2] *Université Côte d'Azur, CNRS, LEAT, France*
[3] *Université Côte d'Azur, INRIA, France*
[4] *Université Côte d'Azur, CNRS, LJAD, France*
elisabetta.de-maria@.unice.fr,lyvonnet@i3s.unice.fr,daniel.gaffe@unice.fr
annie.ressouche@inria.fr,grammont@unice.fr

December 1, 2017

Abstract

In the literature, neuronal networks are often represented as graphs where each node symbolizes a neuron and each arc stands for a synaptic connection. Some specific neuronal graphs have biologically relevant structures and behaviors and we call them archetypes. Six of them have already been characterized and validated using formal methods. In this work, we tackle the next logical step and proceed to the study of the properties of their couplings. For this purpose, we rely on Leaky Integrate and Fire neuron modeling and we use the synchronous programming language Lustre to implement the neuronal archetypes and to formalize their expected properties. Then, we exploit an associated model checker called kind2 to automatically validate these behaviors. We show that, when the archetypes are coupled, either these behaviors are slightly modulated or they give way to a brand new behavior. We can also observe that different archetype couplings can give rise to strictly identical behaviors. Our results show that time coding modeling is more suited than rate coding modeling for this kind of studies.

Keywords— Neuronal Networks, Leaky Integrate and Fire Modeling, Model Coupling, Synchronous Languages, Model Checking.

1 Introduction

In our central nervous system, neurons seem to form some small specific graphs presenting biologically relevant structures and behaviors [2]. These graphs, to which we refer as archetypes, can be coupled to constitute the elementary bricks of bigger neuronal networks. In this work, we formally study the dynamical

behavior of some representative archetypes and their couplings. Archetypes can be seen as the syllables on a given alphabet. The composition of two (or more) syllables can give rise to either existent or non-existent words. In this similitude, archetype coupling can give way to observed biological behaviors or unknown ones. Again, two different syllable compositions may generate two words having the same meaning. At the same way, several archetype couplings can display the same behavior.

As far as neuronal networks are concerned, in the literature their modeling is classified into three generations [14, 16]. First generation models, represented by McCulloch-Pitts one [15], handle discrete inputs and outputs and their computational units consist in a set of logic gates with a threshold activation function. Second generation models, whose most emblematic one is the multi-layer perceptron [1], exploit real valued activation functions. These networks, whose real-valued outputs represent neuron firing rates, are widely used in the domain of artificial intelligence. Third generation networks, also called *Spiking Neural Networks* [16], are characterized by the relevance of time aspects. Precise spike firing times are indeed taken into account. Furthermore, they consider not only current input spikes but also past ones (temporal summation). In [11], Spiking Neural Networks are classified with respect to their biophysical plausibility, that is, to the number of behaviors (i.e., typical responses to an input pattern) they can reproduce. Among these models, the Hodgkin-Huxley model [10] is the one able to reproduce most behaviors. However, its simulation process is really expensive even for a few neurons and for a small amount of time. In this work we choose to use the Leaky Integrate and Fire (LI&F) model [13], a computationally efficient approximation of single-compartment model, which proves to be amenable for formal verification.

Neuronal networks can be considered as reactive systems, that is, systems that constantly interact with the environment and which may have an infinite duration. In fact, neuronal networks can be seen as a sequence of reactions to some stimuli. The *synchronous paradigm* is particularly suited to model this class of systems [8]. It is based on the notion of logical time: time is seen as a discrete sequence of instants and at each instant it is possible to observe some inputs, to make some computations, and to produce some outputs. Several languages implement this paradigm; we choose to use the synchronous Language *Lustre* [6], which well fits to neuronal network encoding. Lustre allows not only to easily model neuronal networks, but also to express some properties concerning their dynamical evolution. This is possible thanks to the principle of *observers*. An observer of a property is a program, taking as inputs the inputs/outputs of the model under verification, and deciding at each instant whether the property is violated or not. The tools automatically checking the property satisfaction/violation are called *model checkers*. In this work we use the model checker *kind2* [5], which is compatible with Lustre, to prove some relevant properties concerning neuronal archetypes and their coupling.

The paper, which is the follow-up of [2], is organized as follows. In Section 2 we present a discrete version of the LI&F model. Section 3 is devoted to the description of the neuronal archetypes we take into consideration. In Section

4 we briefly introduce the Language Lustre and model checking techniques. In Section 5 we show archetypes behaviors (encoded in Lustre) and in Section 6 we deal with their coupling. Some related conclusions and future directions are presented in Section 7. All the presented archetype couplings and relative properties have been implemented and the code can be found at:
`https://redmine.i3s.unice.fr/projects/archetypes/repository`.

2 Discrete Leaky Integrate and Fire Model

In this section, we introduce a discrete (Boolean) version of LI&F modeling. We first present all the biological knowledge associated to the modeled phenomena and then we detail the adopted model.

When a neuron receives a signal at one of its synaptic connections, it produces an excitatory or an inhibitory *post-synaptic potential* (PSP) caused by the opening of selective ion channels according to the post-synaptic receptor nature. An activation leads to an inflow of cations in the cell; an inhibition leads to an inflow of anions in the cell. This local ions flow influences the electrochemical potential difference on both sides of the plasma membrane and locally depolarizes (excitation) or hyper-polarizes (inhibition) the neuron membrane. Such polarization is transmitted, step by step, to the rest of the membrane and thus influences the potential difference on both sides of the membrane at the whole cell level. The potential difference is called *membrane potential*. In general, a PSP alone is not enough for the membrane potential of the receiving neuron to overcome its *depolarization threshold*, and thus to emit an *action potential* at its axon to transmit the signal to other neurons.

However, two phenomena allow the cell to overcome its depolarization threshold : the *spatial summation* and the *temporal summation* [17]. Spatial summation allows to sum all the different PSPs produced at a given time at different areas of the membrane. Temporal summation allows to sum all the different PSPs produced between two close enough instants. This summation can be done thanks to a property of the membrane that behaves like a capacitor and can locally store some electrical loads (*capacitive property*).

The neuron membrane, due to the presence of leakage channels, is not a perfect conductor and can be compared to a resistor inside an electrical circuit. Thus, the range of the PSPs decreases with time and space (*resistivity* of the membrane).

A LI&F neuron network is represented with a weighted directed graph where each node stands for a neuron soma and each edge stands for a synaptic connection between two neurons. The associated weight for each edge is an indicator of the weight of the connection on the receiving neuron: a positive (resp. negative) weight is an activation (resp. inhibition).

The depolarization threshold of each neuron is modeled via the *firing threshold* τ, which is a numerical value that the neuron membrane potential p shall overcome at a given time t to emit an action potential, or *spike*, at the time $t + 1$.

The membrane resistivity is symbolized with a numerical coefficient called *leak factor* r, which allows to decrease the range of a PSP over time.

Spatial summation is implicitly taken into account. In our model, a neuron u is connected to another neuron v via a single synaptic connection of weight w_{uv}. This connection represents the entirety of the shared connections between u and v. Spatial summation is also more explicitly taken into account with the fact that, for each instant, the neuron sums each signal received from each input neuron. To take the temporal summation into account, we ensure that past PSPs are summed to current PSPs. Since the older is a PSP, the more it is decreased by r and the less it impacts the membrane potential calculus, we restrict the past PSPs to a given time interval to decrease computational load for each neuron. We introduce a *sliding integration window* of length σ for each neuron. This allows us to obtain finite state sets, and thus to easily apply model checking techniques.

More formally, the following definition can be given:

Definition 1 (Boolean Spiking Integrate and Fire Neural Network)**.** A spiking Boolean integrate and fire neural network *is a tuple* (V, E, w)*, where:*

- V *are Boolean spiking integrate and fire neurons,*
- $E \subseteq V \times V$ *are synapses,*
- $w : E \to \mathbb{Q} \cap [-1, 1]$ *is the synapse weight function associating to each synapse* (u, v) *a weight* w_{uv}*.*

A spiking Boolean integrate and fire neuron *is a tuple* (τ, r, p, y)*, where:*

- $\tau \in \mathbb{N}$ *is the* firing threshold*,*
- $r \in \mathbb{Q} \cap [0, 1]$ *is the* leak factor*,*
- $p : \mathbb{N} \to \mathbb{Q}_0^+$ *is the [membrane]* potential *function defined as*

$$p(t) = \begin{cases} \sum_{i=1}^{m} w_i \cdot x_i(t), & if\ p(t-1) \geqslant \tau \\ \sum_{i=1}^{m} w_i \cdot x_i(t) + r \cdot p(t-1), & otherwise \end{cases}$$

  where $p(0) = 0$*,* m *is the number of inputs of the neuron,* w_i *is the weight of the synapse connecting the* i^{th} *input neuron to the current neuron, and* $x_i(t) \in \{0, 1\}$ *is the signal received at the time* t *by the neuron through its* i^{th} *input synapse (observe that, after the potential overcomes its threshold, it is reset to* 0*),*

- $y : \mathbb{N} \to \{0, 1\}$ *is the neuron output function, defined as*

$$y(t) = \begin{cases} 1 & \text{if } p(t-1) \geqslant \tau \\ 0 & \text{otherwise.} \end{cases}$$

The development of the recursive equation for the membrane potential function and the introduction of a sliding time window of length σ lead to the following equation for p (when $p(t-1) < \tau$): $p(t) = \sum_{e=0}^{\sigma} r^e \sum_{i=1}^{m} w_i \cdot x_i(t-e)$, where e represents the time elapsed until the current one.

3 The Basic Archetypes

The six basic archetypes we study are the following ones (see Fig. 1). These archetypes can be coupled to constitute the elementary bricks of bigger neuronal circuits.

- **Simple series.** It is a sequence of neurons where each element of the chain receives as input the output of the preceding one.
- **Series with multiple outputs.** It is a series where, at each time unit, we are interested in knowing the outputs of all the neurons (i.e., all the neurons are considered as output ones).
- **Parallel composition.** It is a set of neurons receiving as input the output of a given neuron.
- **Negative loop.** It is a loop consisting of two neurons: the first neuron activates the second one while the latter inhibits the former one.
- **Inhibition of a behavior.** It consists of two neurons, the first one inhibiting the second one.
- **Contralateral inhibition.** It consists of two or more neurons, each one inhibiting the other ones.

4 The synchronous language Lustre and Model Checking

Lustre [6] is a synchronous programming language adopted for the conception and verification of reactive systems. It is characterized by the *synchronicity hypothesis*, which depends on the notion of *logical time*. Time is seen as a sequence of discrete instants, in which systems react to the inputs and immediately generate outputs. A Lustre program is a system of equations defining variables, which are functions from time to their domain of values. Time can be projected onto the set of naturals, making variables infinite sequences of values. Lustre is a functional language operating on *flows*, which are pairs of possibly infinite sequences of values of a given type and *clocks*, representing sequences of instants. A program has a cyclic behavior and, at its nth execution cycle, all the involved flows take their nth values.

The unit in the Lustre language is called a *node* and Lustre nodes compute output variable sequences of values from input variable ones with equations, expressions and assertions. Variables are typed: there are a few elementary basic types (Boolean, integer, real) and complex external types (such as vectors) can be declared. Usual operators over basic types are available: `+, -, ...`; `and, or, not`; `if then else`. These are called data operators and only operate on operands sharing the same clock. They operate point wise on sequences of values

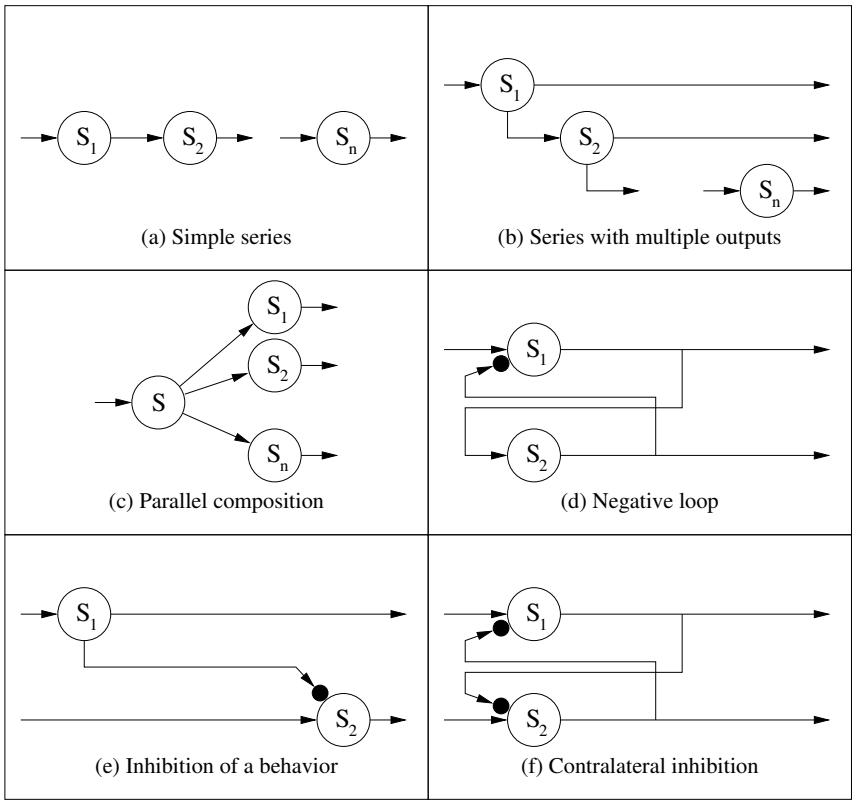


Figure 1: The basic neuronal archetypes.

of their operands. Moreover, Lustre has operators to deal with the logical time represented by clocks. The two main temporal operators are `pre` and $\rightarrow$:

- `pre` (for previous) acts as a memory: if $(e_1, e_2, ..., e_n, \dots)$ is the flow `E`, `pre(E)` is the flow $(nil, e_1, e_2, ..., e_n, \dots)$, nil being an undefined value denoting uninitialized memory.
- $\rightarrow$ (meaning "followed by") complements the `pre` operator and allows to avoid uninitialized memory:
  let `E` $= (e_1, e_2, \dots, e_n, \dots)$ and `F` $= (f_1, f_2, \dots, f_n, \dots)$ be two flows, then `E`$\rightarrow$`F` is the expression $(e_1, f_2, \dots, f_n, \dots)$.

Equations help us to define output variables in nodes. Besides, to force variable values, assertions may complement equations. Assertions consist in Boolean expressions that should be always true and they allow to make assumptions on the environment. For instance, the assertion:
`assert (true` $\rightarrow$ `not(x and pre(x)))` says that `x` is not true twice consecutively in its value sequence.

Lustre is a unifying framework allowing not only to model reactive systems but also to express some temporal properties. Properties are encoded as Lustre nodes called *observers* [7], which check at each instant the outputs of the program under verification and give as a result `true` if the wished behavior is satisfied, `false` otherwise. The automatic verification is made thanks to the tool kind2 [5], selected for his compatibility with Lustre and for his effectiveness with respect to other model checkers such as Lesar [8] and Nbac [12]. In case the desired property is not verified for all possible input variable values, kind2 gives a counterexample, that is, an execution trace showing the violation of the property.

5 Encoding Neuronal Archetypes and Temporal Properties in Lustre

Single neurons can be easily encoded in Lustre. The key idea is to take advantage of an input matrix recording the signals received within the integration window. More precisely, at each time unit the first column of this matrix contains the current inputs (multiplied by the synaptic weights of the corresponding input edges) and, for each `i` greater than 0, the values of the column i are defined as follows: (i) they equal 0 at the first time unit (initialization) and (ii) for all following time units, they are reset to 0 in case of spike emission at the preceding time unit and they take the previous time unit value of the column $i-1$ (for the corresponding row) otherwise. This mechanism, implemented thanks to the use of the `pre` operator, allows to obtain a sliding time window. At each time unit, the input matrix is multiplied by a vector of remaining coefficients and, if the firing threshold is overtaken, a spike is emitted at the next time unit. Thanks to the modularity of Lustre, archetypes can be easily encoded starting from basic neurons.

In the following, we introduce properties of single neurons and archetypes (we denote `true` as 1 and `false` as 0).

5.1 Single neuron

As far a single neuron is concerned, we prove that, whatever its parameters are, it can only display one of the two exclusive following behaviors: either the sum of the current input signals (multiplied by their weights) is enough to overtake the threshold (and thus, for each instant when the neuron receives enough input signals, it emits a spike at the next instant), or more than one time unit is needed to overtake the threshold. More formally, given a neuron with one input flow, property 1 has been proved thanks to kind2:

Property 1. ***[Single neuron.]*** *Given a neuron receiving an input flow on the alphabet* $\{0, 1\}$*, it can only express one of the two exclusive following behaviors:*

Delayer effect. *It emits a* 0 *followed by a flow identical to the input one.*

Filter effect. *It emits at least two* 0 *at the beginning and can never emit two consecutive* 1.[1]

In the first case, the neuron just transmits a sequence identical to the received one, with a delay of one time unit. In the second case, the neuron lets only pass one signal out of x, it thus behaves as a $1/x$ filter. Observe that the filter effect reaches a limit case when the neuron is never able to overtake its threshold (*wall* effect). The Lustre observer of property 1 consists in an exclusive conjunction between a delayer and a filter:

```
node delayer (X: bool; w: int) returns (OK: bool);
    var  SX, Out, S1, verif, preverif: bool;
    let
       Out = neuron(X,w);
       S1 = true  -> Out;
       SX = false  -> pre(X);
       verif = true  -> if SX then S1 else false;
       preverif = true  -> pre(verif)
       OK = if  preverif and verif then true else false;
   tel

node filter (X : bool; w : int) returns (OK : bool)
    var Out : bool;
    let
       Out = neuron (X, w);
       OK = true  -> if Out then not pre (Out) else  not Out;
    tel

node prop1 (X : bool; w : int) returns (OK : bool);
     var S1, S2, Verif : bool;
     let
        S1 = delayer (X, w);
        S2 = filter (X, w);
        Verif = S1 xor S2;
        OK = confirm_property_2_ticks (X) or Verif;
     tel
```

5.2 Simple Series (see Fig. 1(a))

The behavior of a single series of length n depends on the neurons composing it: either it contains n delayers (and in such a case it behaves as delayer of n time units), or it contains at least one filter (it thus displays a n-delayer effect composed of a filter effect). More formally, the following property can be stated:

[1] From a biological point of view, in the first case (delayer) we can speak of instantaneous integrator and in the second case (filter) of long time integrator.

Property 2. ***[n-delayer or n-delayer/filter.]*** *Given a series of length n receiving an input flow on the alphabet $\{0, 1\}$, it can only express one of the two exclusive following behaviors:*

n-delayer effect. *It emits a sequence of* 0 *of length n followed by a flow identical to the input one.*

n-delayer/filter effect. *It emits a sequence of* 0 *of length at least $n + 1$ and can never emit two consecutive* 1*.*

As a consequence, a simple series is not able to constitute a permanent signal, e.g., if it receives an oscillatory signal as input, it cannot emit a sequence of 1 as output. We can also observe that filter neurons do *not commute* in a simple series. As an example, if in a series of two neurons a 1/2 filter (that is, a neuron emitting a 1 every two time units when receiving a permanent signal) precedes a 1/3 filter, the result is a 1/6 filter. If the two neurons are inverted, the result is a wall effect. In order to prevent wall effects, the most selective neurons should thus come first.

5.3 Series with Multiple Outputs (see Fig. 1(b))

As expected, in a series with multiple outputs each emission is constrained by the ones of the preceding neurons. Furthermore, the following property holds:

Property 3. ***[Exclusive temporal activation in a series with multiples outputs.]*** *When a series of n delayers with multiples outputs receives the output of a $1/n$ filter, only one neuron at a time overtakes its threshold (and thus emits a spike).*

As a consequence, in the conditions of Property 3 two neurons can never emit at the same time.

5.4 Parallel Composition (see Fig. 1(c))

Concerning the parallel composition, we saw that the number of spikes emitted in parallel at each instant is in between a given interval, whose upper bound is not necessarily the number of neurons in parallel (because, even if each neuron is able to emit, the parallel neurons can be unsynchronized).

Moreover, we have the following interesting property:

Property 4. ***[Parallel composition of n filters.]*** *Given a parallel composition with a delayer connected to n filters of different selectivity connected to the same delayer, it is possible to emit as output a sequence of* 1 *of length k, with $k \geq n$.*

To obtain a sequence of 1 of length k, the idea is to consider the parallel composition of n filters of different selectivity $1/X_i$, where $i \in \{1, ., n\}$ and X_1,...,X_n is the set of prime numbers in between 2 and k. Furthermore, exploiting a parallel composition, it is possible to constitute a permanent signal from a not permanent one.

5.5 Negative Loop (see Fig. 1(d))

If, neglecting the inhibiting edge, the two neurons of a negative loop behave as delayers, an oscillatory trend can be observed as output of the two neurons (in case a permanent signal is injected in the archetype). More formally:

Property 5. ***[Oscillation in a negative loop.]*** *Given a negative loop composed of two delayers, when a sequence of* 1 *is given as input, the activator neuron oscillates with a pattern of the form* 1100 *(and the inhibitor expresses the same behavior delayed of one time unit).*

5.6 Inhibition of a Behavior (see Fig. 1(e))

For a large class of neuron parameters (that is, firing threshold, leak factor, and length of the integration window), the following property is verified:

Property 6. ***[Fixed point inhibition.]*** *Given an inhibition archetype, if a sequence of* 1 *is given as input, at a certain time the inhibited neuron can only emit* 0 *values.*

5.7 Contralateral Inhibition (see Fig. 1(f))

There is a class of neuron parameters for which the following behavior is displayed as output of contralateral inhibitions:

Property 7. ***[Winner takes all in a contralateral inhibition.]*** *Given a contralateral inhibition archetype with two or more neurons, if a sequence of* 1 *is given as input, starting from a given time one neuron is activated and the other ones are inhibited.*

Some further properties concerning the different archetypes can be found in [3].

6 Archetype Coupling

The two sensible ways to couple two archetypes are the following ones: (i) connect the output(s) of the first archetype to the input(s) of the following one, (ii) nest the first archetype within the second one. In this section we present some emblematic couplings among the ones we studied. In the following figures, these acronyms are employed: A for activator, I for inhibitor, D for delayer, F for filter, G for pattern generator, S for series, and C for collector.

6.1 Simple series within a negative loop

Let us consider a series of n delayers nested between the activator and the inhibitor of a negative loop, as shown in Figure 2.

We remind that, when the negative loop is considered alone and a sequence of 1 is injected in the archetype, an oscillation of the form 1100 is displayed (see

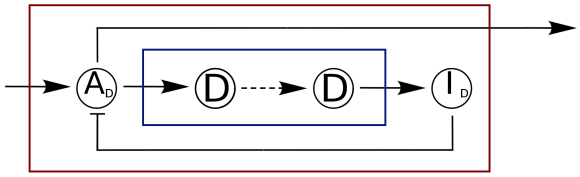


Figure 2: Series of n delayers nested between the activator and the inhibitor of a negative loop.

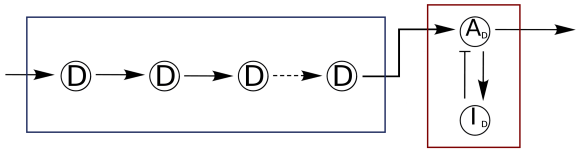


Figure 3: Series of n delayers connected to the activator of a negative loop.

Property 5). The addition of the series results in an extension of the oscillation period, which passes from 2 to n. More precisely:

Property 8. ***[Oscillation period extension.]*** *Given a simple series of delayers of length n within a negative loop, if a sequence of 1 is given as input, the output of the activator is of the form :* $0(1^{n+2}0^{n+2})^{\omega}$. [2]

6.2 Simple series followed by a negative loop

Let us now consider the case in which a series of n delayers precedes the negative loop, as shown in Figure 3.

In this case, the oscillatory behavior is not modified, just delayed. More formally:

Property 9. ***[Oscillation delay.]*** *Given a simple series of delayers of length n connected to the activator of a negative loop, if a sequence of 1 is given as input, the output of the activator is of the form :* $0^n(1100)^{\omega}$.

It seems now relevant to identify all the input patterns of the negative loop allowing to have an output oscillatory trend. Since the series is only able to transmit (or eventually filter) signals, in the next subsection we introduce a small system which is able to generate all the patterns of fixed length on the alphabet $\{0, 1\}$.

[2] x^y denotes the repetition of x for y time units.
x^{ω} denotes the infinite repetition of x.

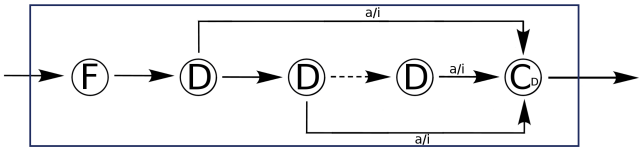


Figure 4: Generator of periodic patterns based on a series with multiple outputs.

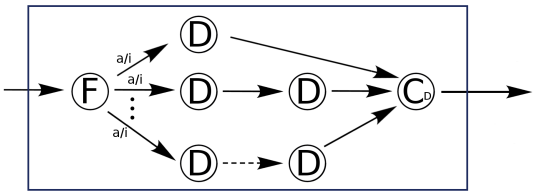


Figure 5: Generator of periodic patterns based on a parallel composition.

6.3 Generator of periodic patterns followed by negative loop

Let us consider the system illustrated in Figure 4. As input of the system, a sequence of 1 is received by a $1/n$ filter connected to a series of n delayers with multiple outputs, which is finally connected to a collector delayer neuron. The user can choose to activate or deactivate each one of the edges connecting the neurons of the series to the collector. We proved that, playing on the possibility of activating/deactivating the aforementioned different edges, such a system can generate all the pattern of length n on the alphabet $\{0, 1\}$.

Another pattern generator is graphically depicted in Figure 5. In this system, a $1/n$ filter is connected to n simple series of delayers, of increasing length from 1 to n. The edges connecting the filter to the series can be modulated, that is, activated or deactivated (the intuition is that the activation of the series of length x allows the emission of a 1 in the x-position of the pattern).

It is possible to see that the first generator has a number of nodes and edges which is linear with respect to the length of the pattern to generate while in the second system the number of nodes and edges is quadratic. For this reason, we rest on the first generator for our studies.

For different values of n, we connected the pattern generator to the entry of the negative loop (see Figure 6) and we detected all the patterns able to produce oscillations (we consider not only strict oscillations, where the number of 1 equals the number of 0, but also oscillations where the difference between the consecutive number of 1 and the consecutive number of 0 is at most 2).

As an example, for $n = 5$, the pattern 11001 generates oscillations of the form 11000 as output of the negative loop (we remind that a simple series receiving

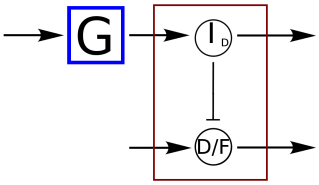


Figure 6: Pattern generator connected to the activator of a negative loop.

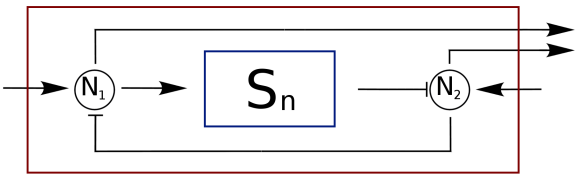


Figure 7: Series of n delayers within a contralateral inhibition of two neurons.

a sequence of 1 as input is not able to emit a pattern of the form 11001).

6.4 Series within a contralateral inhibition

In this subsection we present the insertion of a series of n delayers within a two neurons contralateral inhibition to delay the inhibition of the losing neuron. At this aim, we consider a first neuron N_1 acting as a delayer (if it does not receive any inhibition) connected to a simple series S_n of n delayers, which inhibits a second neuron N_2. The second neuron (which also acts as a delayer if it is not submitted to any inhibition) inhibits N_1 and the first inhibition is stronger than the second one (see Figure 7).

The idea is to study how the winner takes all behavior (see Property 7) is modified when the inhibition of the loser neuron is delayed. We prove that, for several inhibitor edge weights, the insertion of the series makes the system stabilize later. Furthermore, the stabilization is preceded by $n+1$ damped oscillations of period $n+2$ (see Figure 8).

The first oscillation of N_1 is composed of one 1 and a number of 0 equal to $n+1$. For each subsequent oscillation, the number of 1 (resp. of 0) increases (resp. decreases) of one unit. After its last oscillation, N_1 only emits a sequence of 1. The behavior of N_2 is symmetric (its first oscillation is composed of one 0 and a number of 1 equal to $n+1$) .

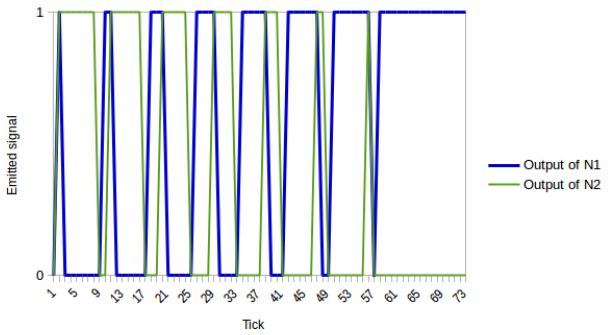


Figure 8: Output of N1 and N2 for a simple series of $n = 6$ delayers nested in a two neurons contralateral inhibition (see Figure 7).

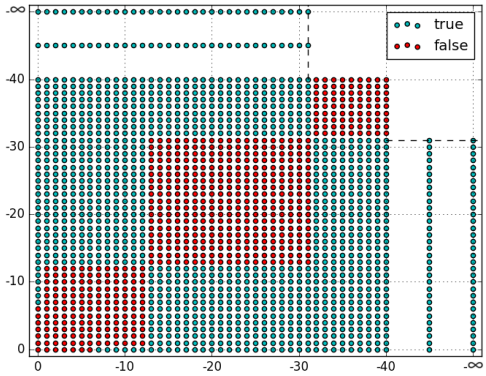
The following property has been encoded in Lustre and validated for several inhibitor edge weights:

Property 10. ***[Winner takes all delay.]*** *Let us consider a delayer neuron N_1 connected to a series of n delayers S_n inhibiting a delayer neuron N_2, which in turn inhibits N_1. The edge inhibiting N_2 has a higher absolute value then the one inhibiting N_1. Let us suppose a sequence of 1 is given as input of the archetype composition. When N_1 emits a sequence of 1 as long as the first sequence of 1 emitted by N_2, then, after the emission of a 0 by N_1, $N1$ (resp. N_2) only emits a sequence of 1 (resp. 0).*

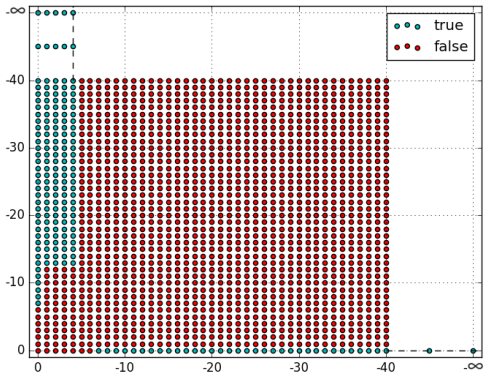
The two diagrams of Figure 9 allow to compare the behavior of a two neurons contralateral inhibition with the one of the same archetype with the insertion of one single delayer. In both plots, the y-axis (resp. x-axis) represents the weight of the edge inhibiting the first (resp. second) neuron. Blue (resp. red) points represent pairs of weight values for which the stabilization is (resp. is not) reached within the first four time units. The passage from the first (a) to the second diagram (b) shows that the red zone (which refers to the non satisfaction of the winner takes all property) increases. Furthermore, the extension of the red zone is asymmetric, which reflects the asymmetry of the composition. It is interesting to see that, in a counter-intuitive way, for the explored weight edge values, the neuron winning more often within the first four time units is the one preceding the series (even if the inhibitor signal it emits is delayed of one time unit).

6.5 Pattern generator followed by an inhibition

Let us finally introduce the pattern generator followed by the inhibition archetype (see Figure 10).



(a)



(b)

Figure 9: Verification of the winner takes all behavior for the different values of the inhibiting weights of the two neurons in a simple contralateral inhibition (a) and in a contralateral inhibition with two neurons and the insertion of a delayer (b).

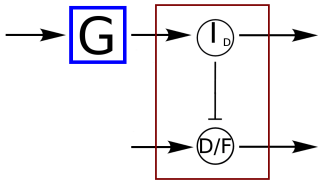


Figure 10: Pattern generator followed by inhibition

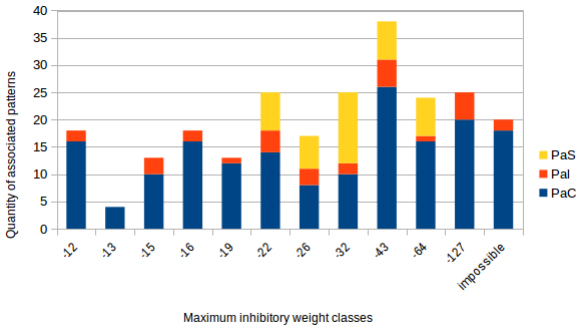


Figure 11: Classes of maximal inhibitory weights allowing the inhibition behavior. The orange color corresponds to palindrome patterns (Pal), the blue color corresponds to patterns whose twin is in the same class (PaC), and the yellow color stands for patterns whose twin falls in another class (PaS).

At the beginning we kept the neuron parameters and edge weights found for the inhibition archetype alone and looked for the input patterns able to provide the wished behavior (that is, starting from a given instant the inhibited neuron stops emitting spikes, see Property 6). With the known parameters, the only pattern allowing to obtain the given behavior is the permanent sequence of 1. We then tried to see whether, strengthening the weight of the inhibiting edge, other patterns give the inhibition behavior. The answer is positive. We tested our property for all patterns of length $n < 8$ and in the histogram of Figure 11 patterns are classified with respect to the maximal inhibitor edge weight allowing to get the wished behavior. The following interesting observation emerges: if a pattern is not palindrome (that is, its reading from the left to the right and from the right to the left does not give the same pattern), it necessarily has a twin pattern (obtained by reading the pattern from the right to the left). There are several cases in which a pattern and his twin one do not fall in the same weight class. As an example, for $n = 7$, the pattern 0110100 and his twin 0010110 are in two different (neighbor) classes. This suggests that in a pattern the proportion of zeros and ones is not the only relevant feature: the way zeros are surrounded by ones is important. Models based on precise *time coding* seem thus more pertinent than models based on *rate coding* for the properties we are interested in.

7 Conclusion and Future Work

In this paper we proposed a formal approach to study some key properties concerning the dynamical evolution of neurons, some canonical archetypes, and their coupling. The synchronous language Lustre turned out to be an unified framework allowing not only to encode neuronal networks but also their expected properties. The properties were automatically verified thanks to the model

checker kind2. Such a tool, which relies on Satisfiability Modulo Theories (SMT) based k-induction, turned out to be more efficient than other model checkers based on convex polyhedra representation of integers and reals [2].

Our results show that archetype coupling can either modulate the behaviors displayed by the single archetypes (e.g. extend an oscillation period) or clearly give rise to new behaviors. Furthermore, several different couplings turn out to display the same behavior (whatever their input sequences are). As a next step, we intend to make a systematic study of all the possible archetypes and couplings, even including several archetypes, and classify their respective properties. The number of constrained graphs of a few elements and, concomitantly, the number of elementary behaviors is reduced, both from a logical and biological standpoint. Moreover, beyond a certain number of neurons and connections, other functional processes like cell assemblies [9] are supposed to emerge, in biological networks and in logical ones if they are set with biologically inspired parameters. So the composition process should stop after a limited number of iterations, that is, we should soon fall back on already studied behaviors, but expressed through different structures.

Before checking the validity of a property, we actually fix an interval of values for each parameter and we ask to the model checker whether the property holds for the chosen parameter set. We are aware of a research group working on the integration of Linear Decision Diagrams (LDDs) within model checkers. The exploitation of LDDs, which consist in decision trees whose leaves are labelled with contraints on the parameters, would allow to automatically infer parameter sets for which properties are verified.

We also intend to translate the Lustre code for neuron archetypes into the VHSIC Hardware Description Language (VHDL) one to make it run on a field-programmable gate array (FPGA) [4], that is, an integrated circuit configurable by the customer. This would allow to have a real physical implementation of our models validated through model checking techniques.

To conclude, our long term aim consist in proving that whatever neuronal circuit can be expressed as a composition of archetypes, as far as all words can be expressed from a given alphabet.

Acknowledgements

This work was founded by the NeuComp project of Academy 1 of University Côte d'Azur. We thank our partners Benoit Miramond and Alexandre Muzy for fruitful discussions.

References

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

[2] Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Verification of Temporal Properties of Neuronal Archetypes Modeled as Synchronous Reactive Systems. In *HSB 2016 - 5th International Workshop Hybrid Systems Biology*, Lecture Notes in Bioinformatics series, page 15, Grenoble, France, October 2016.

[3] Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. Research Report RR-8937, UCA, Inria ; UCA, I3S ; UCA, LEAT ; UCA, LJAD, July 2016.

[4] L. Fiack, B. Miramond, and L. Rodriguez. A neural processing unit for self-organizing maps. In *Workshop on "Neuromorphic and Brain-Based Computing Systems"*, Grenoble, France, Mar 2015.

[5] G. Hagen and C. Tinelli. Scaling up the formal verification of lustre programs with smt-based techniques. In *2008 Formal Methods in Computer-Aided Design*, pages 1–9, Nov 2008.

[6] N. Halbwachs. Synchronous programming of reactive systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998.

[7] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993. Workshops in Computing, Springer Verlag.

[8] N. Halbwachs and P. Raymond. Validation of synchronous reactive systems: from formal verification to automatic testing. In *ASIAN'99, Asian Computing Science Conference*, Phuket (Thailand), December 1999. LNCS 1742, Springer Verlag.

[9] D. O. Hebb. Organization of behavior. new york: Wiley. *The Journal of Physiology*, 6(307):335, 1949.

[10] 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.

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

[12] B. Jeannet. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. *Formal Methods in System Design*, 23(1):5–37, 2003.

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

[14] W. Maas. Networks of spiking neurons: The third generation of neural network models. *Trans. Soc. Comput. Simul. Int.*, 14(4):1659–1671, December 1997.

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

[16] H. Paugam-Moisy and S. M. Bohte. Computing with spiking neuron networks. In *Handbook of Natural Computing*, pages 335–376. 2012.

[17] D. Purves, G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.S. LaMantia, J. O. McNamara, and S. M. Williams, editors. *Neuroscience*. Sinauer Associates, Inc., 3rd edition, 2006.