LI&F neuron: Series and Negative Loops

Model Checking

Fissore Davide

Université Côte d'Azur

Jan. 06, 2023

MASTER INFORMATIQUE

Objective

1. *Implémenter avec le model checker probabiliste PRISM un neurone biologique de type LI&F.*

Objective

1. *Implémenter avec le model checker probabiliste PRISM un neurone biologique de type LI&F.*
2. *Choisir deux parmi les mini-circuits proposés dans la figure 1 de l'article FCS.pdf, les implémenter en PRISM et tester des propriétés de logique temporelle concernant ces mini-circuits.*

The LI&F neuron

Definition (LI&F: Leaky Integrate and Fire Model)

A neuronal network represented by a digraph.

- Nodes represent the neurons
- Edges (the synaptic connections) have either positive (activators) or negative (inhibitors) weights
- Nodes contain a membrane potential: if its threshold is overcome, then a spike is emitted
- The leak factor reduces at each time unit the neuron potential

Some existing archetypes

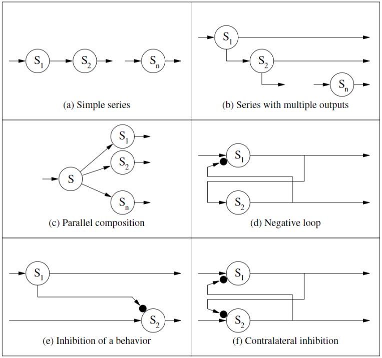


Fig. 1 The basic neuronal archetypes.

[1] Img. from *On the Use of Formal Methods to Model and Verify Neuronal Archetypes*

Some existing archetypes

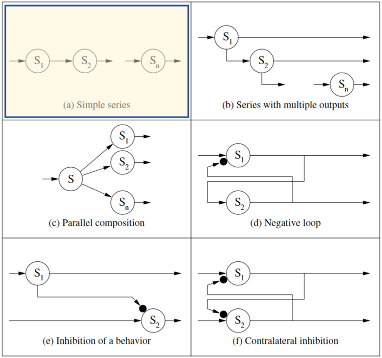


Fig. 1 The basic neuronal archetypes.

[1] Img. from *On the Use of Formal Methods to Model and Verify Neuronal Archetypes*

Some existing archetypes

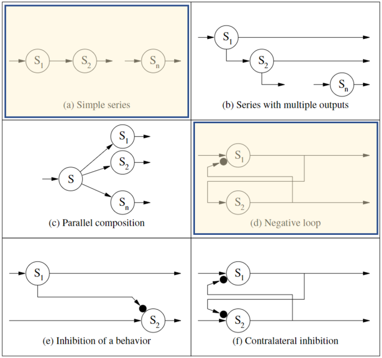


Fig. 1 The basic neuronal archetypes.

[1] Img. from *On the Use of Formal Methods to Model and Verify Neuronal Archetypes*

The simple serie archetype

Definition (Simple serie archetype)

A series of n neurons $N_1, ..., N_n$, where N_i receives the signal form N_{i-1} and emits its signal to N_{i+1}.

There exists 2 main implementations of this archetype:

- the n-delayer
- the n-delayer/filter

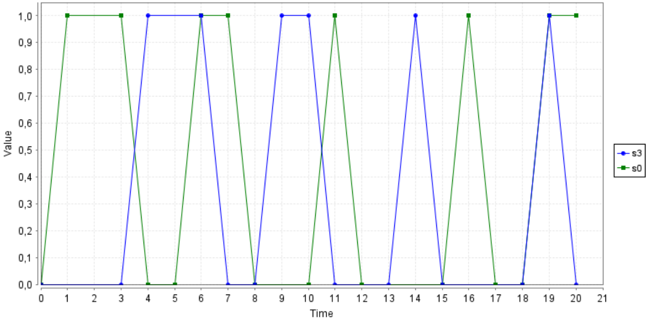
The n-delayer

Given a signal $\mathcal{S}$, $\mathcal{S}$ is transmitted with a delay of n time units.

Implementation:

```
dtmc

module S3 = S1 [s1=s3, s0=s2] endmodule

module S2 = S1 [s1=s2, s0=s1] endmodule

module S1
        s1 : [0..1] init 0;
        [s] true -> (s1'=s0);
endmodule

module initialisation
        s0 : [0..1] init 0;
        [s] true -> 0.5:(s0'=1) + 0.5:(s0'=0);
        //[s] true -> (s0'=1-s0); // alternating sequence
endmodule
```

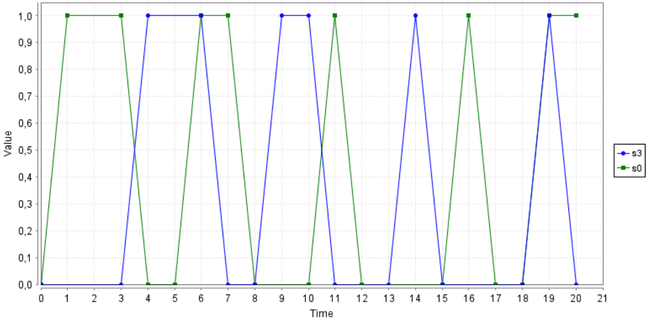
A plot of a n-delayer



An interesting property:

- $P = [G(s_0 = 1 \Leftrightarrow X^3(s_3 = 1))]$?

A plot of a n-delayer



An interesting property:

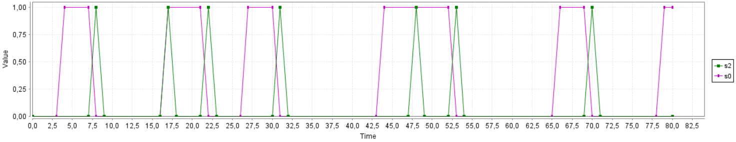
- $P = [G(s_0 = 1 \Leftrightarrow X^3(s_3 = 1))]?\ 1.0$

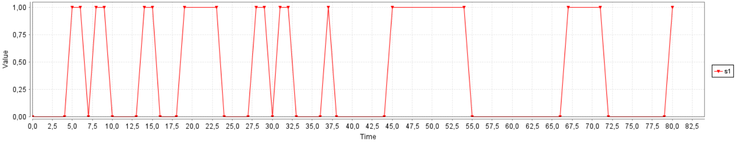
The n-delayer/filter

Let's introduce the leak factor and the potential membrane: An example of **code corrector**.

```
dtmc
const double noise = 0.1;
const int rep = 3; // nb of digit repetition

module S2
        pot2: [0..10]  init 0;
        s2 : [0..1];
        [s] s2=0 & cnt < rep -> (pot2'=ceil(pot2*0.9+s1)) & (s2'=0);
        [s] s2=0 & cnt = rep & pot2 > 1.5 -> (s2'=1) & (pot2'=0);
        [s] s2=0 & cnt = rep & pot2 <= 1.5 -> (pot2'=0);
        [reset] s2=1 -> (s2'=0) & (pot2'=0);
endmodule

module S1
        s1 : [0..1] init 0;
        [s] true -> noise:(s1'=1-s0) + (1-noise):(s1'=s0);
endmodule

module initialisation
        cnt: [0..rep] init 0;
        s0 : [0..1] init 0;
        [s] cnt < rep -> (cnt'=cnt+1);
        [s] cnt = rep -> 0.5:(s0'=0) + 0.5:(s0'=1) & (cnt'=0);
        //[s] cnt = rep -> (s0'=1-s0) & (cnt'=0);
endmodule
```

A plot of a n-delayer/filter





Test certainty of model by noise

$$P \stackrel{?}{=} \begin{cases} (X^4((s_0 = 1) \Leftrightarrow X^5(s_2 = 1))) & \text{if } noise > 0.5 \\ (X^4((s_0 = 1) \Leftrightarrow X^4(s_2 = 1))) & \text{otherwise} \end{cases}$$

noise	proba
0.00	1.00
0.10	0.90
0.50	0.50
0.90	0.18
1.00	0.00

Remark

If the $noise$ is bigger than 0.5, we have to look at the fifth time-unit: one time-unit is consumed by the $reset$ action.

The negative loop archetype

Definition (Negative loop archetype)

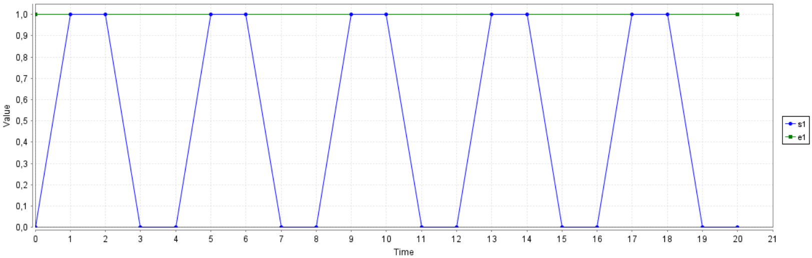
Two neurons where the first receives the input signal and an inhibition from the second and the second is activated by the first.

```
dtmc

const int wS1 = 1;
const int wS2 = -1;

module S2
        s2 : [0..1] init 0;
        [s] true -> (s2' = wS1 * s1);
endmodule

module S1
        s1 : [0..1] init 0;
        [s] true -> (s1' = max(0, e1 + wS2 * s2));
endmodule

module E1
        e1 : [0..1] init 1;
        //[s] true -> 0.5:(e1'=1) + 0.5:(e1'=0);
        [s] true -> (e1'=1);
endmodule
```

Remark

Note the max function → the signal of $S1$ can only be 0 or 1.

A plot of negative loop



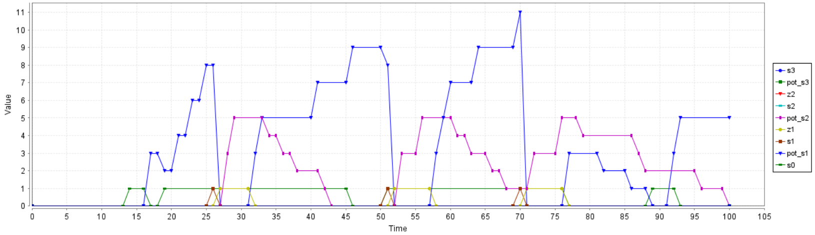
Negative loop in LI&F style

```
dtmc

const int wS1 = 1;
const int wS2 = -1;

module S2
        s2 : [0..1] init 0;
        [s2] true -> 0.8:(s2' = wS1 * z1) + 0.2:(s2' = 1 - (wS1 * z1));
endmodule

module transf1_2
        z1: [0..1] init 0;
        [to1] true -> (z1'=0);
        [reset1] true -> (z1'=1);
endmodule

formula pot_upd = max(0, ceil(pot * 0.9) + e1 + wS2 * s2);

module S1
        s1 : [0..1] init 0;
        pot : [0..100] init 0;
        [to1] s1 = 0 & pot <= 1  -> 0.2:(s1'=1) + 0.8:(s1'=0) & (pot' = pot_upd);
        [to1] s1 = 0 & pot <= 2 & pot > 1  -> 0.5:(s1'=1) + 0.5:(s1'=0) & (pot' = pot_upd);
        [to1] s1 = 0 & pot > 2 -> (pot' = pot_upd) & (s1'=1);
        [reset1] s1 = 1 -> (pot' = 0) & (s1'=0);
endmodule

module E1
        e1 : [0..1] init 1;
        //[s] true -> 0.5:(e1'=1) + 0.5:(e1'=0);
        [to1] true -> (e1'=1);
endmodule
```

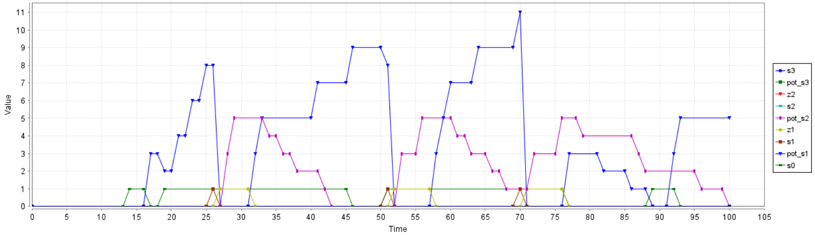
A complete example

```
..
module S3 = S1 [s1=s3, pot_s1=pot_s3, s0=z2, threshold_s1=threshold_s3, to1=to3, reset1=reset3] endmodule

module transf2_3 = transf1_2 [z1=z2, to1=to2, reset1=reset2] endmodule

module S2 = S1 [s1=s2, pot_s1=pot_s2, s0=z1, threshold_s1=threshold_s2, to1=to2, reset1=reset2] endmodule

module transf1_2
        z1: [0..1] init 0;
        [to1] true -> (z1'=0);
        [reset1] true -> (z1'=1);
endmodule

module S1

    s1 : [0..1] init 0;     // 0 = inactif et 1 = spike
    pot_s1 : [0..m] init 0; // potential of the neuron

    [to1] s1=0 & pot_s1 <= threshold_s1/2 ->
          (pot_s1'=floor(pot_s1 * fuite + s0 * power));
    [to1] s1=0 & pot_s1 > threshold_s1/2 & pot_s1 <= threshold_s1 ->
          0.2:(s1'=1) + 0.8:(s1'=0) & (pot_s1'=floor(pot_s1 * fuite + s0 * power));
    [to1] s1=0 & pot_s1 > threshold_s1 ->
          (s1'=1) & (pot_s1'=floor(pot_s1 * fuite + s0 * power));
    [reset1] s1=1 -> (s1' = 0) & (pot_s1' = 0);

endmodule

module initialisation
        s0 : [0..1] init 0;
        [to1] s0=0 -> 1:(s0' = 1);
        [to1] s0=1 -> 1:(s0' = 0);
endmodule
```

A corresponding plot



A corresponding plot



Manual exploration

Module/[action]	Probability	Update
[to3]	0.3333333333333333	pot_s3'=0
[reset2]	0.3333333333333333	z2'=1, s2'=0, pot_s2'=0
[to1]	0.1666666666666666	z1'=0, pot_s1'=0, s0'=1
[to1]	0.1666666666666666	z1'=0, pot_s1'=0, s0'=0

☑ Generate time automatically

Conclusion

Programming strategy:

- Implement simple and deterministic models
- Introduce probabilities
- Add the leak factor along with the potential membrane

Conclusion

Programming strategy:

- Implement simple and deterministic models
- Introduce probabilities
- Add the leak factor along with the potential membrane

About the project:

- Understand the logic behind neuron interactions
- Implement neurons in PRISM
- Test concretely how do they work

Thanks