

LI&F neuron: Series and Negative Loops

Model Checking

Fissore Davide

Université Côte d'Azur

Jan. 06, 2023



MASTER
INFORMATIQUE

- 1 Introduction
- 2 The Simple Series archetype
- 3 The negative loop archetype
- 4 Conclusion

The LI&F neuron

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

A neuronal network represented by a digraph.

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

Leaky Integrate and Fire Model

Some existing archetypes

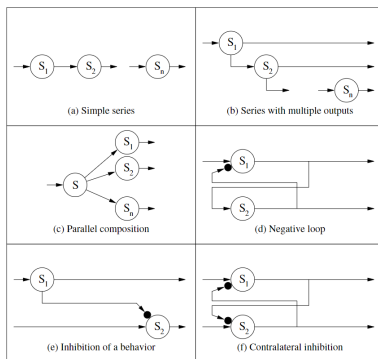


Fig. 1 The basic neuronal archetypes.

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

Some existing archetypes

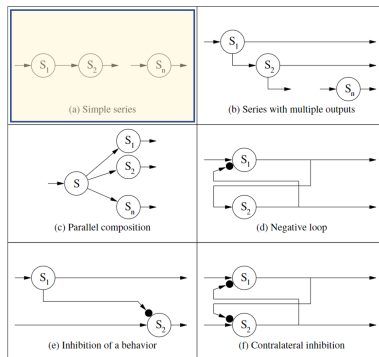


Fig. 1 The basic neuronal archetypes.

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

Some existing archetypes

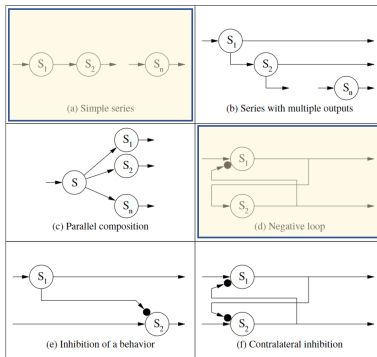


Fig. 1 The basic neuronal archetypes.

¹Img. 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, \dots, N_n , where N_i receives the signal from N_{i-1} and emits its signal to N_{i+1} .

There exists 2 main implementations of this archetype:

- the n -layer
- the n -layer/filter

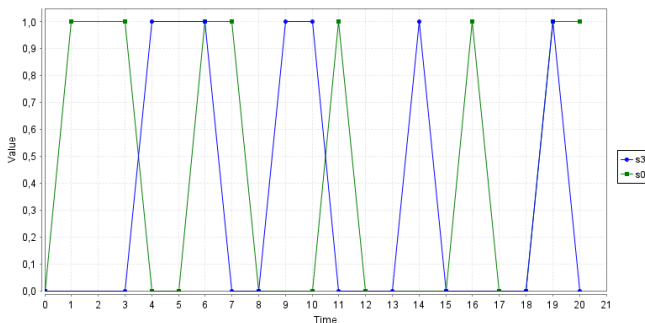
The n -delayer

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

Simple implementation:

```
1 dtmc
2
3 module S3 = S1 [s1=s3, s0=s2] endmodule
4
5 module S2 = S1 [s1=s2, s0=s1] endmodule
6
7 module S1
8     s1 : [0..1] init 0;
9     [s] true -> (s1'=s0);
10 endmodule
11
12 module initialisation
13     s0 : [0..1] init 0;
14     [s] true -> 0.5: (s0'=1) + 0.5: (s0'=0);
15     // [s] true -> (s0'=1-s0); // alternating sequence
16 endmodule
```

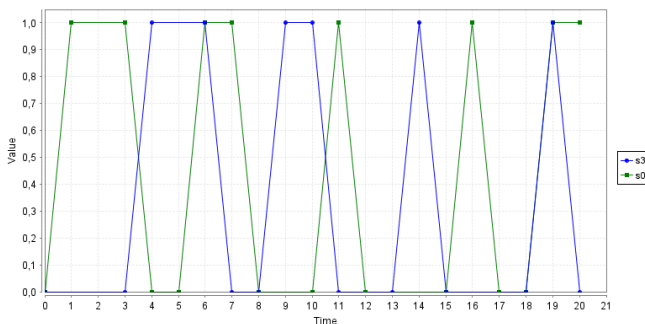

A plot of a n -delayer



An interesting property:

$$\bullet P \stackrel{?}{=} [G(s_0 = 1 \Leftrightarrow (X^3 s_3 = 1))]$$

A plot of a n -delayer



An interesting property:

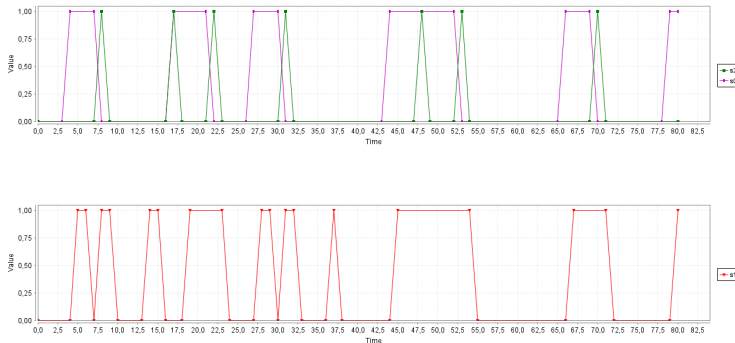
● $P \stackrel{?}{=} [G(s_0 = 1 \Leftrightarrow (X^3 s_3 = 1))]$ Answer: 1.0

The n -delayer/filter

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

```
1 dtmc
2 const double noise = 0.1;
3 const int rep = 3; // nb of digit repetition
4
5 module S2
6     pot2: [0..10] init 0;
7     s2 : [0..1];
8     [s] s2=0 & cnt < rep -> (pot2'=ceil(pot2*0.9+s1)) & (s2'=0);
9     [s] s2=0 & cnt = rep & pot2 > 1.5 -> (s2'=1) & (pot2'=0);
10    [s] s2=0 & cnt = rep & pot2 <= 1.5 -> (pot2'=0);
11    [reset] s2=1 -> (s2'=0) & (pot2'=0);
12 endmodule
13
14 module S1
15     s1 : [0..1] init 0;
16     [s] true -> noise:(s1'=1-s0) + (1-noise):(s1'=s0);
17 endmodule
18
19 module initialisation
20     cnt: [0..rep] init 0;
21     s0 : [0..1] init 0;
22     [s] cnt < rep -> (cnt'=cnt+1);
23     [s] cnt = rep -> 0.5:(s0'=0) + 0.5:(s0'=1) & (cnt'=0);
24     //[s] cnt = rep -> (s0'=1-s0) & (cnt'=0);
25 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.

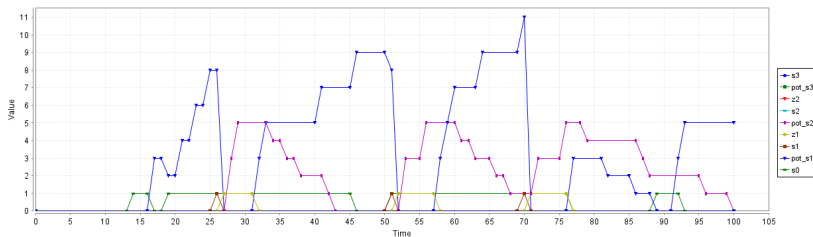
A complete example

```

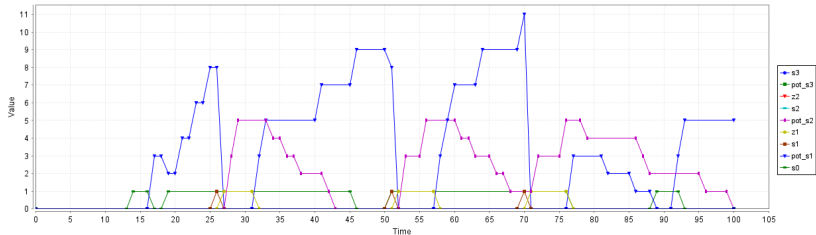
10 ..
11 module S3 = S1 [s1=s3, pot_s1=pot_s3, s0=z2, threshold_s1-threshold_s3, tol1=tol3, reset1=reset3] endmodule
12
13 module transf2_3 = transf1_2 [z1=z2, tol1=tol2, reset1=reset2] endmodule
14
15 module S2 = S1 [s1=s2, pot_s1=pot_s2, s0=z1, threshold_s1-threshold_s2, tol1=tol2, reset1=reset2] endmodule
16
17 module transf1_2
18     z1: [0..1] init 0;
19     [tol] true -> (z1'=0);
20     [reset1] true -> (z1'=1);
21 endmodule
22
23 module S1
24
25     s1: [0..1] init 0; // 0 = inactif et 1 = spike
26     pot_s1: [0..m] init 0; // potential of the neuron
27
28     [tol] s1=0 & pot_s1 <= threshold_s1/2 ->
29         (pot_s1'=floor(pot_s1 + fuite + s0 * power));
30     [tol] s1=0 & pot_s1 > threshold_s1/2 & pot_s1 <= threshold_s1 ->
31         0.2:(s1'=1) + 0.8:(s1'=0) & (pot_s1'=floor(pot_s1 + fuite + s0 * power));
32     [tol] s1=0 & pot_s1 > threshold_s1 ->
33         (s1'=1) & (pot_s1'=floor(pot_s1 + fuite + s0 * power));
34     [reset1] s1=1 -> (s1' = 0) & (pot_s1' = 0);
35
36 endmodule
37
38 module initialisation
39     s0: [0..1] init 0;
40     [tol] s0=0 -> 1:(s0' = 1);
41     [tol] s0=1 -> 1:(s0' = 0);
42 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

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.

```

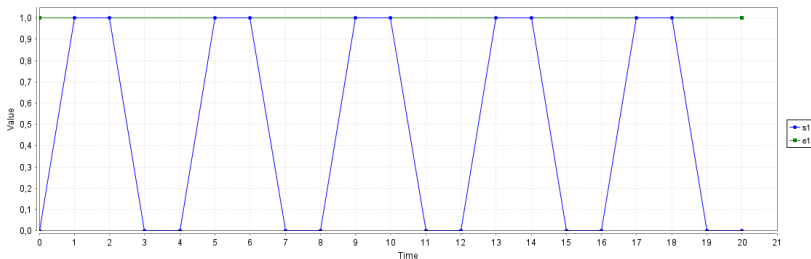
1 dtmc
2
3 const int wS1 = 1;
4 const int wS2 = -1;
5
6 module S2
7     s2 : [0..1] init 0;
8     [s] true -> {s2' = wS1 * s1};
9 endmodule
10
11 module S1
12     s1 : [0..1] init 0;
13     [s] true -> {s1' = max(0, e1 + wS2 * s2)};
14 endmodule
15
16 module E1
17     e1 : [0..1] init 1;
18     //[s] true -> 0.5:(e1'-1) + 0.5:(e1'-0);
19     [s] true -> {e1'=1};
20 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

```

1 dtmc
2
3 const int wS1 = 1;
4 const int wS2 = -1;
5
6 module S2
7     s2 : [0..1] init 0;
8     [s2] true -> 0.8:(s2' = wS1 * z1) + 0.2:(s2' = 1 - (wS1 * z1));
9 endmodule
10
11 module transf1_2
12     z1 : [0..1] init 0;
13     [tol] true -> (z1'=0);
14     [reset1] true -> (z1'=-1);
15 endmodule
16
17 formula pot_upd = max(0, ceil(pot * 0.9) + e1 + wS2 * s2);
18
19 module S1
20     s1 : [0..1] init 0;
21     pot : [0..100] init 0;
22     [tol] s1 = 0 & pot <= 1 -> 0.2:(s1'=1) + 0.8:(s1'=0) & (pot' = pot_upd);
23     [tol] s1 = 0 & pot <= 2 & pot > 1 -> 0.5:(s1'=1) + 0.5:(s1'=0) & (pot' = pot_upd);
24     [tol] s1 = 0 & pot > 2 -> (pot' = pot_upd) & (s1'=1);
25     [reset1] s1 = 1 -> (pot' = 0) & (s1'=0);
26 endmodule
27
28 module E1
29     e1 : [0..1] init 1;
30     //[s] true -> 0.5:(e1'=1) + 0.5:(e1'=0);
31     [tol] true -> (e1'=1);
32 endmodule

```

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