Prof. dr. ing. Tiberiu-Ștefan Leția

As. Drd. Ing. Dahlia Al-Janabi

2020

Synthesis of a controller for reactive applications

MOLDOVAN OVIDIU

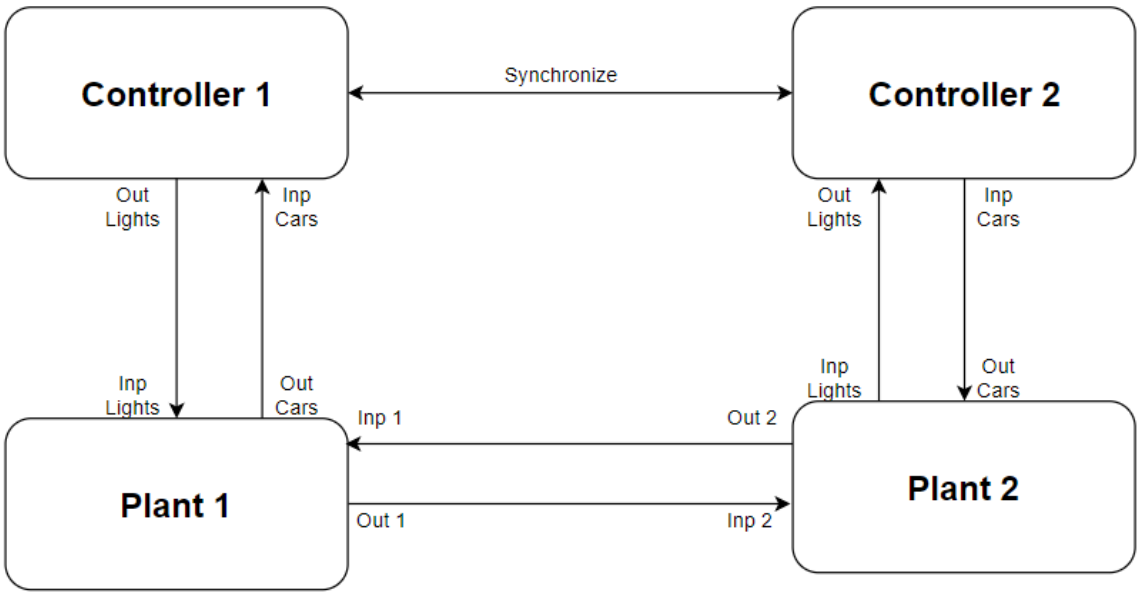
NENU MARIA

MUNTE VLAD LIVIU GROUP 30341

TABLE OF CONTENT

THE SYNTHESIS PROBLEM

The specifications are based on the $OETPN_p$ model of the plant, used as a generic name here. In our case the plants represent different traffic intersections with traffic lights for each lane, while the controllers are used to compute the switch between lights. The OETPN model also describes interactions between lanes and intersections.



For the plant, we specified:

- Inp_1 input channels receive the number of cars that leave the second intersection and enter the first one.
- Out_1 output channels give the number of cars that leave the first intersection and enter the second one.
- Out_2 output channels give the number of cars that leave the second intersection and enter the first one.
- Inp_2 input channels receive the number of cars that leave the first intersection and enter the second one.

We have been given the following intersections: (city: Sibiu, around the RAMADA hotel)

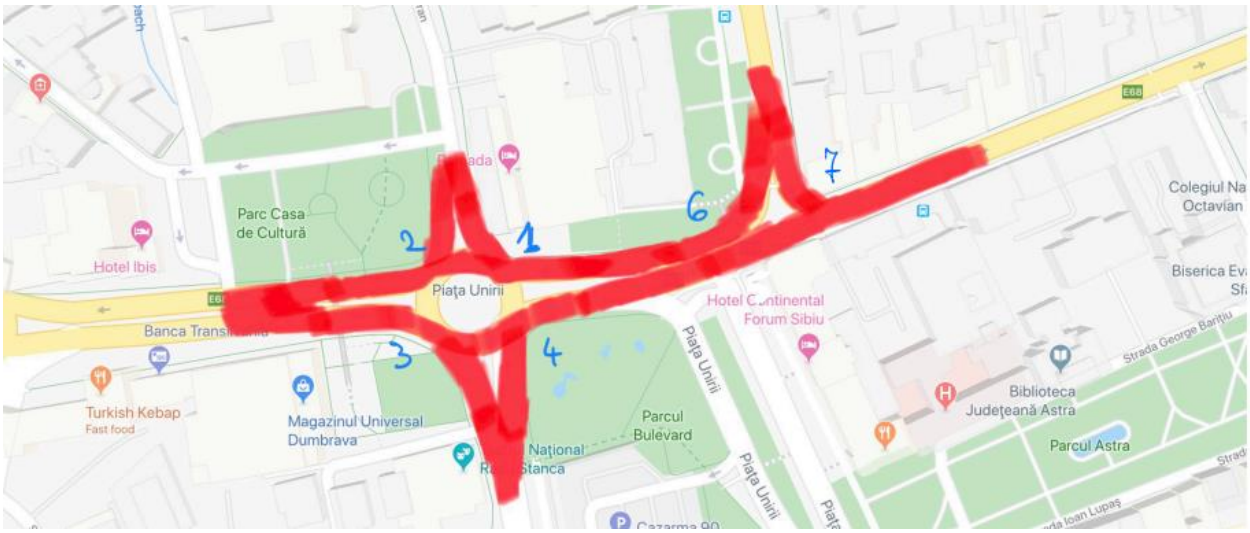


Figure 1 – Google Maps image of the intersections

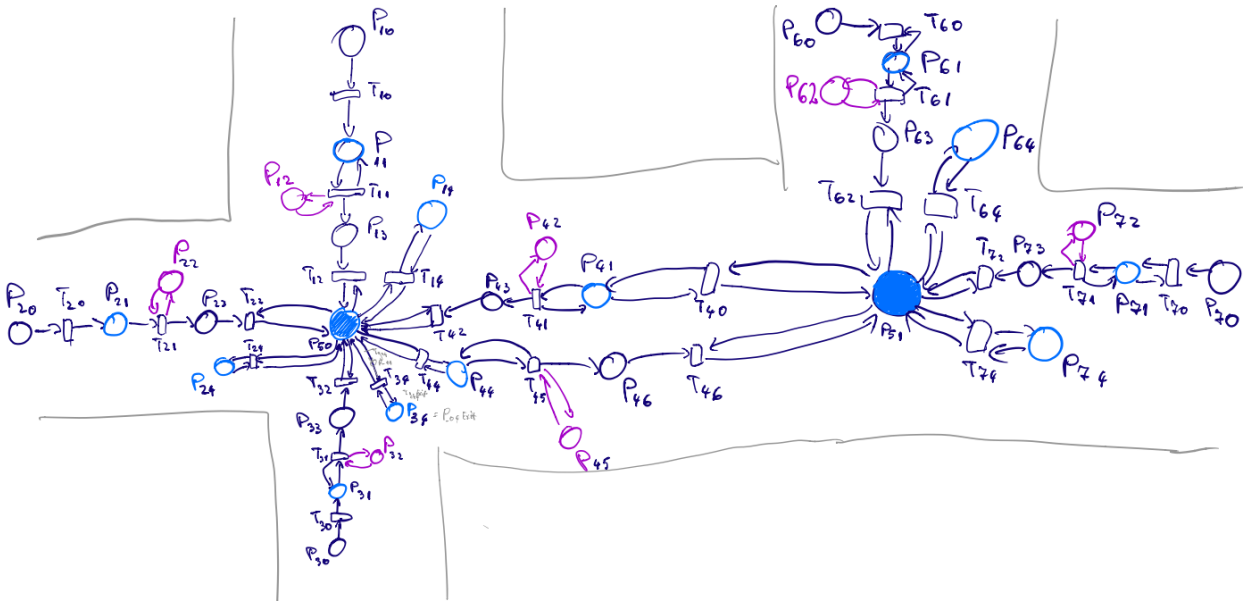


Figure 2 – Lanes in Petri Nets

**Px1 – input, Px2 – traffic light, Px4 - output*

Figure 2 is our model for the lanes with Petri Nets.

For the first intersection, we assumed that instead of the roundabout, we would have a 4 lanes intersection with our implementation of the controller. (*Figure 3*, *ControllerProject1.java* -> controller with 4 phases)

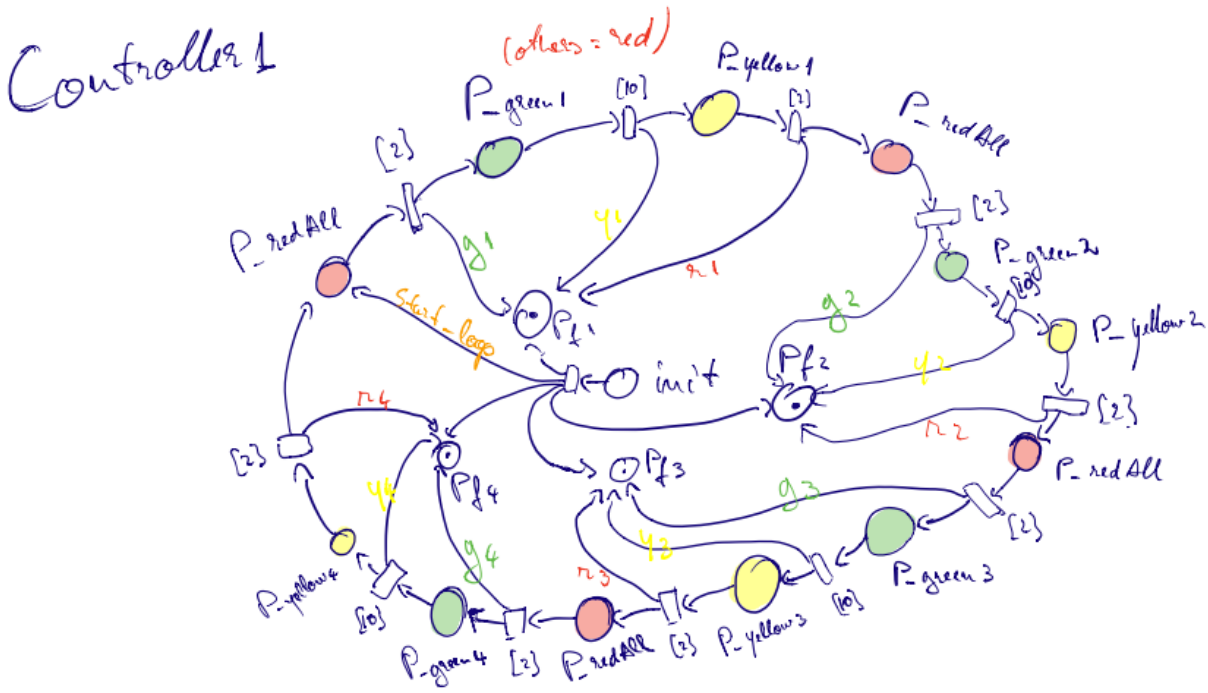


Figure 3 – Controller 1 – 4 phases controller

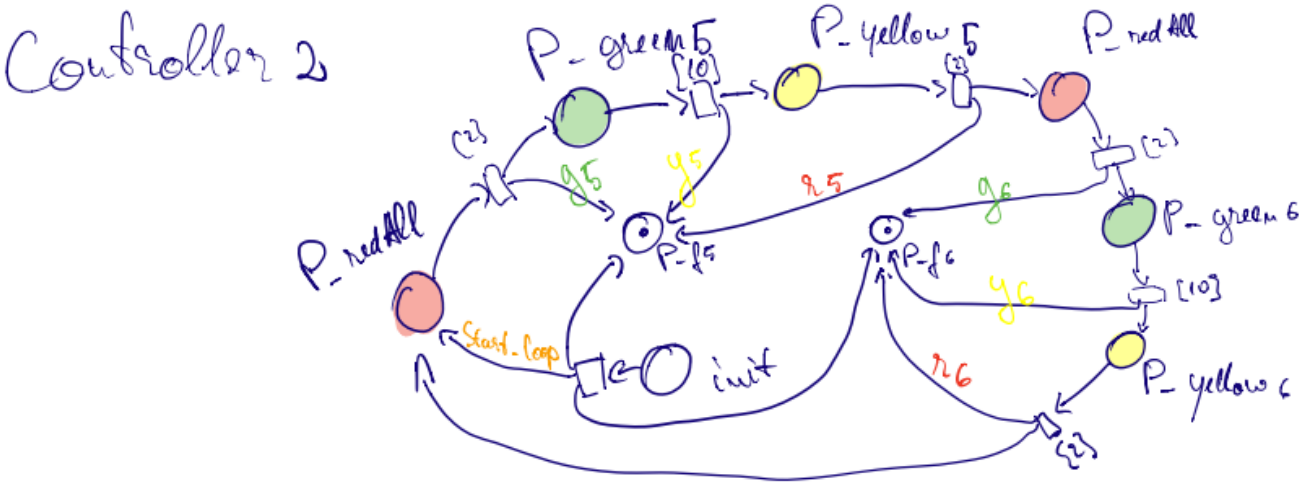


Figure 4 – Controller 2 – 2 phases controller

For the second intersection, we thought at first of implementing 3 traffic lights (*Figure 5*, *ControllerProject3.java* -> controller with 3 phases) but then we took a closer look and 2 of the streets are one-way streets. With that in mind, we designed the controller with only 2 traffic lights, accordingly *(Figure 4*, *ControllerProject2.java* -> controller with 2 phases).

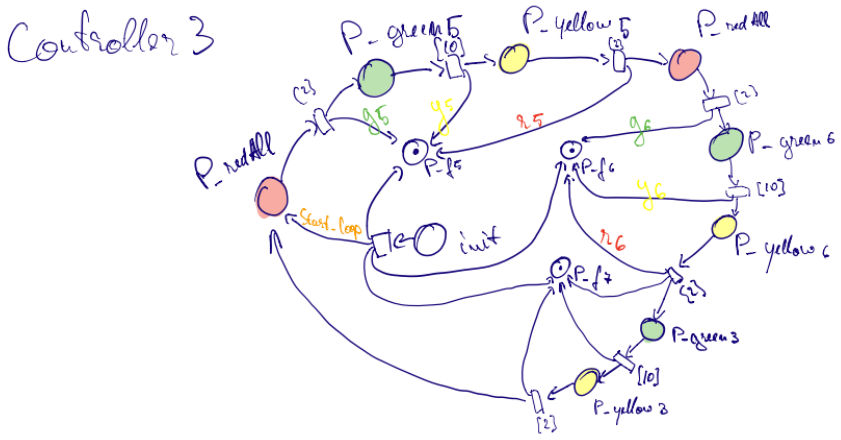
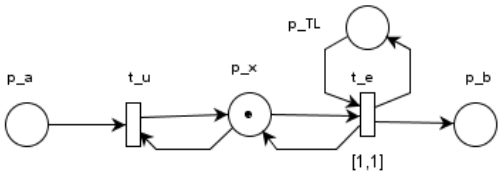


Figure 5 – Controller 3 – 3 phases controller

EVENT-BASED PLANT MODEL

The model of a lane is: $x_c = x_a + u - e$



Where x_c and x_a represents the number of cars on the current and previous lane respectively, and u and e are the events when the entry / exit of a car is signaled. u and e have the value one, only when the respective sensor signals the passage of the machine, and the rest have zero value. The event model of the lane is represented in the attached figure.

Notations:

- p_a the input channel from the input sensor α_i,
- p_b the output channel from the output sensor β_i,
- t_u the transition that takes over the input event and increases the number of cars x on the lane stored in p_x
- p_TL the input channel through which TL commands are received (r, y, g)
- t_e the transition that models (and determines) the output of a car from the lane

The Place types are:

- type(p_a) = InputEvent
- type(p_x) = integer
- type(p_TL) = char = {r, y, g}
- type(p_b) = OutputEvent = {forward, right; f, r}

The evolution rules associated with the transitions are:

- t_u: grd: m(p_a) ≠ ϕ; map_u: x=x+1

t_e: grd: m(p_TL) = g; map_e: m(p_b) = event_f or event_r; random()

Intersection model:

The paths of the open lanes in a phase cannot intersect. The system has two phases marked with: phase_1 (for IL_1 and IL_3) and phase_2 (for IL_2 and IL_4).

The attached model describes the behavior of the intersection.

In our implementation we represented:

- Px1 – input
- Px2 – traffic light
- Px4 – output

The place types are:

- type(p_b1) = type(p_b2) = type(p_b3)=type(p_b4)= Event(f, r);
- type(p_o1) = type(p_o2) = type(p_o3) = type(p_o4) = Event
- type(p_I) = ListVector with a list for each output lane.

CHECKING IF THERE ARE CARS IN A LANE

For checking if there is a car in a lane before turning that traffic light, in the lanes model, we define the auxiliary locations and an “isCar” DataString like in the example below:

```
DataTransfer p_aux1 = new DataTransfer();
p_aux1.SetName("Yu11");
p_aux1.Value = new TransferOperation("localhost", "1081", "Y11");
pn.PlaceList.add(p_aux1);

DataString isCar = new DataString();
isCar.SetName("isCar");
isCar.SetValue("isCar");
pn.PlaceList.add(isCar);
```

Then, we sent this information to the controller:

```
grdT1.Activations.add(new Activation(t10, "isCar", TransitionOperation.SendOverNetwork, "Yu11"));
grdT1.Activations.add(new Activation(t10, "isCar", TransitionOperation.Move, "isCar"));
```

On the controller side, the DataStrings received by the lane is defined:

```
DataString p14 = new DataString();
p14.SetName("Y11");
pn.PlaceList.add(p14);
```

We create a new condition to check if there is a car and bind it to the previous condition using AND logic:

```
Condition T1Ct2 = new Condition(t1, "Y11", TransitionCondition.NotNull);

T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);
```

In the end, we have to add transitions to the states so that the controller changes the check in each lanes. Otherwise, the controller will not move until a car comes in lane 1. This is done in fig. :

```
PetriTransition t11 = new PetriTransition(pn);
t11.TransitionName = "T11";
t11.InputPlaceName.add("y1r2r3r4");
t11.InputPlaceName.add("Y12");
t11.InputPlaceName.add("red");
t11.InputPlaceName.add("yellow");



Condition T11Ct1 = new Condition(t11, "y1r2r3r4", TransitionCondition.NotNull);
Condition T11Ct2 = new Condition(t11, "Y12", TransitionCondition.IsNull);

T11Ct1.SetNextCondition(LogicConnector.AND, T11Ct2); //token in both locations so that a car should be in the lane

GuardMapping grdT11 = new GuardMapping();
grdT11.condition= T11Ct1;
grdT11.Activations.add(new Activation(t11, "y1r2r3r4", TransitionOperation.Move, "r1y2r3r4"));
grdT11.Activations.add(new Activation(t11, "red", TransitionOperation.SendOverNetwork, "OP1"));
grdT11.Activations.add(new Activation(t11, "red", TransitionOperation.SendOverNetwork, "OP3"));
grdT11.Activations.add(new Activation(t11, "yellow", TransitionOperation.SendOverNetwork, "OP2"));
grdT11.Activations.add(new Activation(t11, "red", TransitionOperation.SendOverNetwork, "OP4"));

grdT11.Activations.add(new Activation(t11, "red", TransitionOperation.Move, "red"));
grdT11.Activations.add(new Activation(t11, "yellow", TransitionOperation.Move, "yellow"));

t11.GuardMappingList.add(grdT11);

t11.Delay = 5;
pn.Transitions.add(t11);
```

The full implementation is available in the appendix.

APPENDIX

Because the size of the code was way too big to be put here, you can find it all on the following Github link:

https://github.com/ovi1602/TrafficLightsControllers

The source files we have worked on the most can be found on the following links:

Controller 1 – controller in 4 phases

https://github.com/ovi1602/TrafficLightsControllers/blob/master/src/Test/ControllerProject1.java

Controller 2 – controller in 2 phases

https://github.com/ovi1602/TrafficLightsControllers/blob/master/src/Test/ControllerProject2.java

Controller 3 – controller in 3 phases

https://github.com/ovi1602/TrafficLightsControllers/blob/master/src/Test/ControllerProject3.java

Lanes Intersection

https://github.com/ovi1602/TrafficLightsControllers/blob/master/src/Test/Lanes_Intersection.java