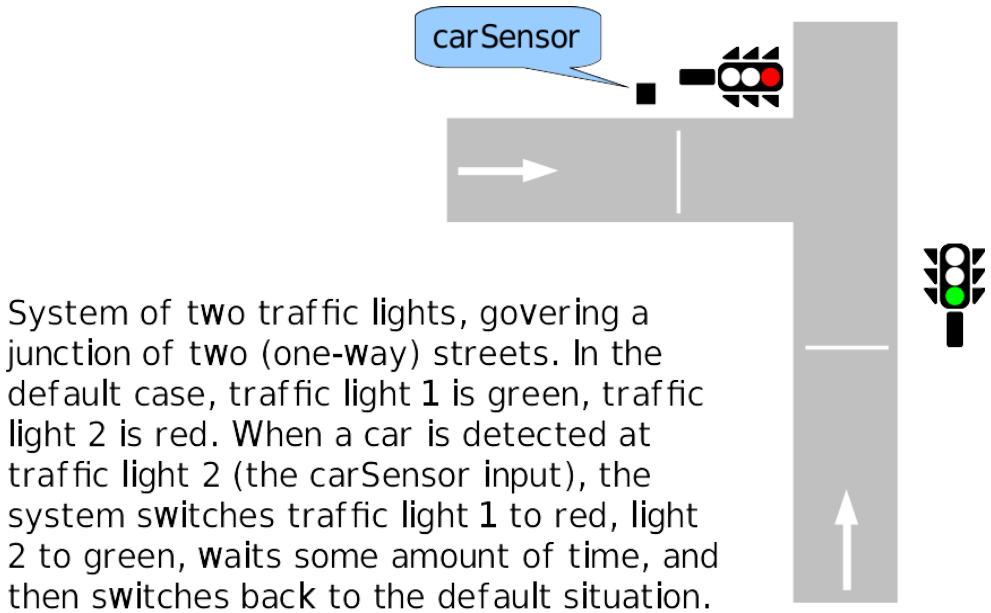
Model-Based Design of Embedded Systems: Lustre Lab

Friday 2020-10-23, 13:15 - 17:00

Background

The purpose of this lab is to practise implementation, specification, and verification of reactive systems in the language Lustre. The lab will continue the work on the traffic light system, which was already discussed in the lecture on 2020-10-14:



System of two traffic lights, govering a junction of two (one-way) streets. In the default case, traffic light 1 is green, traffic light 2 is red. When a car is detected at traffic light 2 (the carSensor input), the system switches traffic light 1 to red, light 2 to green, waits some amount of time, and then switches back to the default situation.

We started implementing this system in the lecture, and came up with the following code outline:

```
-- One traffice light
node TrafficLight(initialRed, toGreen, toRed : bool)
      returns (green, amber, red : bool);
let
  - idea: use a counter!
  green = ...;
  amber = ...;
  red   = ...;
tel

-- Traffic light system
```

```
node TSystem(carSensor : bool) returns (g1, a1, r1, g2, a2, r2 : bool)
let
  (g1, a1, r1) = TrafficLight(false, ...); -- light 1
  (g2, a2, r2) = TrafficLight(true,  ...); -- light 2
tel
```

Your task in the lab will be to finish the implementation, specify some of the important safety properties of the system, and verify the correctness of the system with respect to those properties using Luke or Kind2.

Task 1: Implementation

Finish the implementation according to the description provided above. As inspiration, you can have a look at the (simpler) traffic system provided as an example on the Kind2 webpage;[1] you can use a similar approach in your `TrafficLight` node to correctly manage the change between the signals green, amber, and red.

Hints and points to take into account in the implementation:

- You will need to introduce further local variables in the nodes.
- You can use all Lustre nodes that were introduced in the lectures.
- Our traffic lights are supposed to follow the European standard, which means that red and amber should be lit at the same time while switching from red to green.
- Make reasonable choices for the delays when switching between green and red, and vice versa, and for the time traffic light 2 will show green before switching back to red.

Carefully test your implementation using the Luke/Kind2 simulator.

Task 2: Specification

Since traffic lights are safety-critical, we want to specify some of the most important properties formally. For this we follow the process described in the lecture: we first formulate such properties in natural language, and then translate them to Lustre expressions. The translated requirements are put into a synchronous observer.

In this task, we will provide some of the properties in natural language, and you should translate them to Lustre; you are also supposed to come up with some further safety properties that complement the ones given by us, and translate those to Lustre. You can use the temporal nodes introduced in the lecture for the specification.

`TrafficLight` Node

- **R1:** It cannot happen that red and green are lit at the same time.

[1] https://kind.cs.uiowa.edu/app/#examples%2FTrafficLight

- **R2:** At least one of the lights (red, amber, green) is always lit.
- **R3:** *(your own property)*

```
-- Synchronous observer for the traffic light node
node ReqTrafficLight(initialRed, toGreen, toRed : bool)
returns (ok1, ok2, ok3 : bool);
var green, amber, red : bool;
let
  (green,  amber,  red) = TrafficLight(initRed,  toGreen, toRed);
  -- no red and green at the same time
  ok1 = <add Lustre expression>;
  -- some signal is always shown
  ok2 = <add Lustre expression>;
  -- <add your own safety property>
  ok3 = <add Lustre expression>;

  --%PROPERTY ok1;
  --%PROPERTY ok2;
  --%PROPERTY ok3;
tel
```

`TSystem` Node

- **R4:** It cannot happen that both signals are showing green at the same time.
- **R5:** *(your own property*

```
node ReqTSystem(carSensor : bool) returns (ok1, ok2 : bool);
var g1, a1, r1, g2, a2, r2 : bool;
let
  (g1, a1, r1, g2, a2, r2) = TSystem(carSensor);

  -- the signals do not both show green at the same time
  ok1 = <add Lustre expression>;
  -- <add your own safety property>
  ok2 = <add Lustre expression>;

  --%PROPERTY ok1;
  --%PROPERTY ok2;
tel
```

Task 3: Verification

We now want to establish that the implementation from Task 1 satisfies the properties from Task 2. For this, apply the Luke or Kind2 tool in verification mode. If you observe any discrepancies between the implementation and the properties (written as Lustre expressions), investigate

which part is at fault, and correct. The final goal is to have a fully verified implementation, with respect to the properties considered in Task 2.

Submission Instructions

The laboration is carried out in groups of 2 students. Each group should produce a document with

- The lustre code for the traffic light system
- Specifications (five requirements) in Lustre
- The results by Kind2 (or Luke) when verifying the requirements.

You do not need to produce text to describe your solution, but the Lustre code should be readable, with comments where appropriate. The document should be uploaded on Studentportalen (as Assignment 4) by Sunday, Oct. 25.

Further Material

- Recorded lecture(s) on Lustre: https://www.youtube.com/playlist?list=PLh7fhnmwbNnzWBfaC8U36mUxZy95J6tz0
- A tutorial of Lustre: http://www-verimag.imag.fr/~halbwach/PS/tutorial.ps
- The standard paper on Lustre: N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. “The synchronous dataflow programming language LUSTRE”
- Luke binaries: http://www.it.uu.se/edu/course/homepage/pins/vt12/lustre
- Kind2 web interface: https://kind.cs.uiowa.edu/app/