Models of Software Systems 2023/24

Raul Barbosa <rbarbosa@dei.uc.pt>

Assignment 2 – Modeling and verifying a fast road intersection

Abstract

This assignment consists in specifying, modeling and verifying an algorithm for controlling a 3-way intersection using traffic lights. Safety should be guaranteed by preventing cars from entering the shared part of the intersection simultaneously from different directions. Furthermore, any car should be able to cross the intersection immediately if there are no other cars attempting to do so. Correctness properties shall be formalized and verified using the SPIN model checker. The assignment shall be carried out in groups of two students.

1 Specifying and modeling a road intersection

Consider the road intersection depicted in Figure 1. There are three traffic lights, containing only *green* and *red* lights, controlled by a computer placed near to the road (typically known as a *roadside unit*). There are also four sensors that detect the presence of vehicles (one sensor in each of the zones).

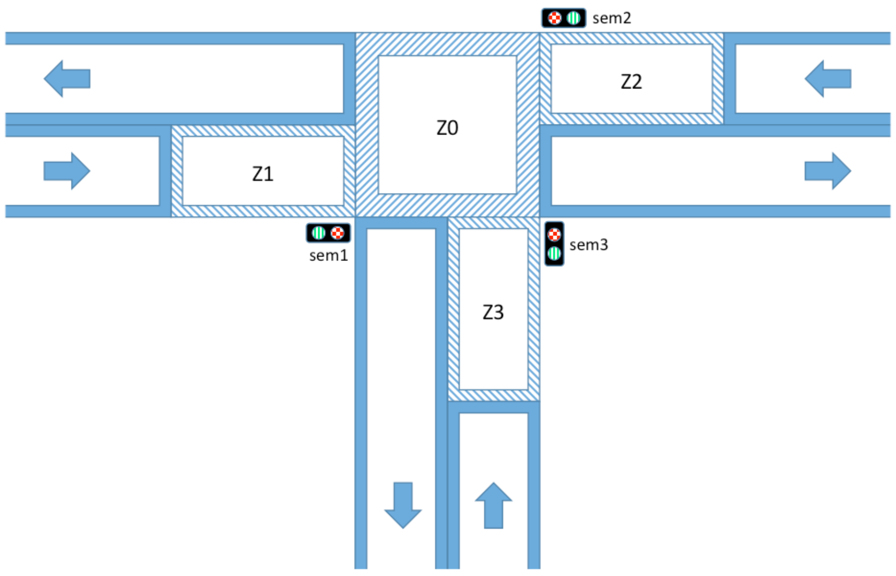

Figure 1: A 3-way road intersection with traffic lights

It is assumed that all vehicles respect traffic rules. The four sensors, named `s0`, `s1`, `s2` and `s3`, are binary (T/F) and only indicate if there is any vehicle, respectively, in zones Z0, Z1, Z2 and Z3. The PROMELA code that follows is an incomplete model of this system.

```
mtype = { red, green }

mtype sem1 = red, sem2 = red, sem3 = red;    // three semaphores
bool  s0, s1, s2, s3;                        // four presence sensors
byte  countZ0, countZ1, countZ2, countZ3;    // count the vehicles in each zone

proctype Sensors() {
  do
  :: s0 = (countZ0 > 0)
  :: s1 = (countZ1 > 0)
  :: s2 = (countZ2 > 0)
  :: s3 = (countZ3 > 0)
  od;
}

proctype RoadsideUnit() {
  do
  :: // update sem1, sem2 and sem3 according to s0, s1, s2 and s3
  od;
}

proctype Vehicles() {
  do
  :: // non-deterministically move vehicles through zones Z0, Z1, Z2, and Z3
  od;
}

init {
  run Sensors();
  run RoadsideUnit();
  run Vehicles();
}
```

The PROMELA code shown above should be regarded as a starting point, although students are free to make any changes deemed necessary. Such changes should be realistic with respect to any assumptions regarding the actual physical intersection.

Part 1. Model the roadside unit such that there may never be simultaneously in Z0 cars coming in from different directions. Whenever there are cars coming in from multiple ways, then at least one of them will get green light eventually. Furthermore, if there are only vehicles coming in from one zone, then those vehicles get a green light as soon as feasible.

You should present a PROMELA model for `proctype RoadsideUnit()` considering that the shared zone Z0 may never have cars coming from more than one way. This is a safety property that may be specified in several ways. Furthermore, if there are some cars arriving in the intersection, eventually one of them shall get green light. This is a liveness property. Additionally, the intersection should be fast and provide the green light as soon as feasible to a car if there are only cars coming in from one of the ways (*i.e.,* cars should not wait unless there is a reason for it). This is also a liveness property.

Notice that the model of the roadside unit should only read the current state from variables `sem1`, `sem2`, `sem3`, `s0`, `s1`, `s2` and `s3`. At each step, it should only write into variables `sem1`, `sem2` and `sem3` in order to update the state.

One is allowed to declare additional variables, *e.g.*, for temporary values or ghost variables, that may be both readable and writable (any such additional variables must be thoroughly justified in the final report). The key is that any such variables must not add unrealistic assumptions to the model.

> **Part 2.** Formalize the correctness properties as LTL formulæ and if needed assertions, (in)valid end states, *etc.*

At least three different properties are needed. First, there may never be cars coming from different directions simultaneously in the shared zone Z0. Second, if there is a car in Z1 then `sem1` will eventually turn green and likewise for Z2 and Z3. Third, if there are only cars coming in from Z1 then `sem1` turns green before any of the other traffic lights turn green again and likewise for Z2 and Z3.

> **Part 3.** Complete the model of the system, namely by writing the PROMELA code for `proctype Vehicles()`.

The behavior of vehicles should be modeled as non-deterministic, updating the values of variables `countZ0`, `countZ1`, `countZ2` and `countZ3`. You should consider that, at any moment, a vehicle may arrive to zones Z1, Z2 and Z3. Furthermore, it is only possible for a vehicle to move into Z0 from Z1, Z2 and Z3 if, respectively, traffic lights `sem1`, `sem2` and `sem3` have the green light turned on. Any vehicle in Z0 may leave the system without any restriction. Notice that one instance of `proctype Vehicles()` is intended to model all the vehicles in the system.

> **Part 4.** Use SPIN to verify the correctness properties and describe the results.

Models should be parametrized to consider up to *N* vehicles simultaneously present in the intersection (obtained by adding all counts). The intention is to limit complexity. Students should examine how the number of states and the verification time grows for increasing values of *N*, and aim to verify with a reasonable value within a reasonable amount of time.

2 Recommended approach and reporting

The final report shall consist of a text file (acceptable formats include plain `.txt` files or, at most, `.md` or `.tex`) and a `.pml` file written to solve the assignment. The written report shall clearly describe the modeling and verification steps taken to address all parts of the assignment. It should also specify the necessary switches passed to all components: SPIN, C compiler and `pan` verifier.

Informally describe the algorithms or protocols that compose the system. Describe also how the complete system is modeled in PROMELA and how correctness properties are formalized from the natural language specification. Arguments must be rigorous even if informal when claiming that the models are coherent with the actual system.

When writing your code bare in mind that it should be readable and understandable by others, just like any programming language. The translation from the idealized system to a language such as PROMELA should be natural and intuitive. The same applies to the specification of correctness properties. Furthermore, aim for writing the smallest sufficient model and avoid unnecessary complexity.