Appendix to Lecture 22: A Fair Traffic Lights System

J. Meseguer

1 Traffic Lights Made Fair

The TRAFFIC-LIGTS system presented in Lecture 24 was a high-level design that left unresolved several issues and their implementation details (this is common in system design). In particular, TRAFFIC-LIGTS allowed *undersired behaviors* where the green light in a given direction (and therefore the red light in the orthogonal diection) could remain on forever. The present appendix presents an easy refinement of the TRAFFIC-LIGTS specification into a more detailed TRAFFIC-LIGTS-FAIR system where *fairness is achieved by design*. The key idea is quite simple: each street direction has a car sensor with a counter that counts how many times cars go across the intersection; and there is a bound on the number of such car passages after which the green light in that direction must turn yellow (it could also have done so earlier, before the bound was reached). Here is the Maude specification, as well as the model checking commands verifying that TRAFFIC-LIGTS-FAIR satisfies the main safety and liveness requirements that one would expect to hold in a traffic lights system. Since such requirements were already discussed in Lecture 24, no futher comments are given, except for short comments describing the meaning of each verified formula(s).

```
mod TRAFFIC-LIGHTS-FAIR is protecting NAT .
  sorts Conf LightState Intersection Direction Light Car Counter .
  subsorts LightState Intersection Car Counter < Conf .
  op mt : -> Conf [ctor] .
  op _ _ : Conf Conf -> Conf [ctor assoc comm id: mt] .
  op [_] : Conf -> Intersection [ctor] .
  ops h v : -> Direction [ctor] .
  op car : Direction -> Car [ctor] .
  ops green red yellow  : Direction -> Light [ctor] .
  op {_,_} : Light Light -> LightState [comm] .
  op cnt : Direction Nat -> Counter .

   op init : -> Conf .

  vars d d1 d2 : Direction . var L : Light .  var n : Nat .

  eq init = {green(h),red(v)} [mt] cnt(h,0) cnt(v,0) .  var C : Conf .

  rl [g2y] : {green(d1),red(d2)} [C] cnt(d1,n) =>
                                  {yellow(d1),red(d2)} [C] cnt(d1,0).
  rl [y2r] : {yellow(d1),red(d2)} [mt]  => {red(d1),green(d2)} [mt] .
```

```
  crl [car.in] : {green(d),L} [mt] cnt(d,n) =>
                     {green(d),L} [car(d)] cnt(d,s(n)) if n < 50 .
crl [car.in] : {green(d),L} [mt] cnt(d,n) =>
              {green(d),L} [car(d) car(d)] cnt(d,s(n)) if n < 50 .
  rl [car.out] : {green(d),L} [car(d) car(d)] => {green(d),L} [mt] .
  rl [car.out] : {green(d),L} [car(d)] => {green(d),L} [mt] .
  rl [car.out] : {yellow(d),L} [car(d) car(d)] => {yellow(d),L} [mt] .
  rl [car.out] : {yellow(d),L} [car(d)] => {yellow(d),L} [mt] .
endm

mod TRAFFIC-LIGHTS-FAIR-PREDS is
  protecting TRAFFIC-LIGHTS-FAIR .
  protecting SATISFACTION .

  subsort Conf < State .

   vars L L' : Light .  vars C C' : Conf .  vars d d1 d2 : Direction .
   vars n m : Nat .

   op enabled : -> Prop .

   eq {green(d1),red(d2)} [C] cnt(d1,n) C' |= enabled = true .
   eq {yellow(d1),red(d2)} [mt] C |= enabled = true .
   ceq {green(d),L} [mt] cnt(d,n) C |= enabled = true if n < 50 .
   eq {green(d),L} [car(d) car(d)] C |= enabled = true .
   eq {green(d),L} [car(d)] C |= enabled = true .
   eq {yellow(d),L} [car(d) car(d)] C |= enabled = true .
   eq {yellow(d),L} [car(d)] C |= enabled = true .

   op on : Light -> Prop [ctor] .

   eq {L,L'} C |= on(L) = true .

   op side-collision-dngr : -> Prop [ctor] .

   eq [car(h) car(v) C'] C |= side-collision-dngr = true .

   op yellow-enabled : Direction -> Prop [ctor] .

   eq {green(d1),red(d2)} [C] cnt(d1,n) C' |= yellow-enabled(d1) = true .
endm


mod TRAFFIC-LIGHTS-FAIR-CHECK is
  protecting TRAFFIC-LIGHTS-FAIR-PREDS .
  including MODEL-CHECKER .

  op yellow-fair : -> Formula .

  eq yellow-fair = (([] <> yellow-enabled(h)) ->  ([] <> on(yellow(h)))) /\
                   (([] <> yellow-enabled(v)) ->  ([] <> on(yellow(v)))) .
endm

*** The system is safe:
```

```
red modelCheck(init,[] ~ side-collision-dngr) .

result Bool: true

*** The system is deadlock-free:

red modelCheck(init,[] enabled) .

result Bool: true

*** The system satisfies the yellow-fair property:

red modelCheck(init,yellow-fair) .

result Bool: true

*** Yellow lights are now on infinitely often:

red modelCheck(init,([] <> on(yellow(h)))) .

result Bool: true

red modelCheck(init,([] <> on(yellow(v)))) .

result Bool: true

*** Green lights are now on infinitely often:

red modelCheck(init,([] <> on(green(v)))) .

result Bool: true

red modelCheck(init,([] <> on(green(h)))) .

result Bool: true

*** Red lights are now on infintely often:

red modelCheck(init,([] <> on(red(v)))) .

result Bool: true

red modelCheck(init,([] <> on(red(h)))) .

result Bool: true

***Green always holds until yellow:

red modelCheck(init,[] (on(green(h)) -> (on(green(h)) U on(yellow(h))))) .

result Bool: true

red modelCheck(init,[] (on(green(v)) -> (on(green(v)) U on(yellow(v))))) .
```

```
result Bool: true

*** Yellow always holds until red:

red modelCheck(init,[] (on(yellow(h)) -> (on(yellow(h)) U on(red(h))))) .

result Bool: true

red modelCheck(init,[] (on(yellow(v)) -> (on(yellow(v)) U on(red(v))))) .

result Bool: true

*** Red always holds until green:

red modelCheck(init,[] (on(red(h)) -> (on(red(h)) U on(green(h))))) .

result Bool: true

red modelCheck(init,[] (on(red(v)) -> (on(red(v)) U on(green(v))))) .

result Bool: true
```

2 From Traffice Lights to Cyber-Physical Systems

Traffice Light systems are an example of so-called *cyber-physical systems* that, besides computing (their "cyber" side), interact with their physical environment, where both space and time play crucial roles. This suggests that a suitable design framework where both time and space can be reflected, which could be used to model in a more expressive way traffic light systems, is that of *real-time rewrite theories* [2] and the associated *Real-Time Maude* specification and verification tool [3], which is available through the Maude web page. In fact, a sophisticated *family* of traffic light system designs have been both specified and verified in Real-Time Maude in [4]. That paper can be a usefull further reading on how time can be used to specify and verify in a more expressive way traffic light systems and other cyber-physical systems. For another example of a sophisticated cyber-physical system also designed and verified in Real-Time Maude, namely, the distributed control system for the turning maneuvers of an airplane, in which both the control laws and the aeorodynamic differential equations were modeled, see [1]. Explicitly modeling time, which is actually used in many distributed systems anyway (e.g., for timeouts), tends to greatly simplify fairness issues, since time rules out many unfair excutions.

References

[1] K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky. Designing and verifying distributed cyber-physical systems using multirate PALS: an airplane turning control system case study. *Sci. Comput. Program.*, 103:13–50, 2015.

[2] P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285:359–405, 2002.

[3] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1–2):161–196, 2007.

[4] P. C. Ölveczky and J. Meseguer. Specification and verification of distributed embedded systems: A traffic intersection product family. In P. C. Ölveczky, editor, *Proceedings First International Workshop on Rewriting Techniques for Real-Time Systems, RTRTS 2010, Longyearbyen, Norway, April 6-9, 2010*, volume 36 of *EPTCS*, pages 137–157, 2010.