Sheil Mehta

CMPT 477 - Modelling Communicating Mobile Devices using NuSMV.

Introduction.

The goal of this project is to model and verify a simple protocol for mobile devices. Using NuSMV, I created three different models: A simple 2 transmitter, 1 car model, a 1 car, 3 transmitter model, and finally a model with 2 cars, 2 transmitters.

I made a few assumptions, and also simplified some aspects of the system, in order to model the system successfully.

Methodology.

I designed three modules, not including the main. I had a module for the transmitters, cars, and the central control. I used different process threads, to account for the different cars/ transmitters. The transmitter was controlled by the car, and the control. Control told the transmitter when to be in the state ‘switch’ which was a when the car switched from using one transmitter to the next. The car was connected to one of the transmitters, through which it could either talk, or switch to another one to talk through.

I made a few assumptions, one of them being that if transmitter was in ‘talk’ state, the car couldn’t switch transmitters and vice versa.

When it came to debugging, using the (SPEC) CTL properties, I was able to detect that my code had a number of bugs. One of them was that both transmitters were on at the same time, even though the car was only connected to one of them. Another bug was that the car was ‘talking’ as well as ‘switching’ transmitters at the same time. CTL also helped with checking if the program was running as intended, with the cars, control and transmitters all behaving in the correct way.

CTL properties and FAIRNESS constraints elaborated below.

-- specification (AG (switch = TRUE & transmitter = tr1) -> AX transmitter = tr2) IN ctrl is true

AG (switch ^ transmitter= 1) -> AX (transmitter= 2)

For all paths globally, when switch and transmitter 1 occur, it is followed by a change to transmitter 2.

-- specification (AG (switch = TRUE & transmitter = tr2) -> AX transmitter = tr1) IN ctrl is true

Same as above for the case of a switch from transmitter 2 to transmitter 1.

-- specification EG switch = TRUE IN ctrl is true

EG (switch)

There exists a path globally, where switch is equal to true. I used this to ensure that switch does eventually exist as true in the system.

-- specification AG !(switch = talk) IN cr is true

For all paths globally, switch is never equally to talk. I included this specification, to make sure that a transmitter was never both in the talk state, as well as the switching state.

-- specification AF (curr = name -> AX stat = TRUE) IN trans1 is true

AF(name -> stat)

All paths, finally if the current transmitter is equal to the name of the transmitter, then it implies that the transmitter is on.

-- specification AF (curr = name -> AX stat = TRUE) IN trans2 is true

(Same as above).

-- specification (AG (switch = TRUE & transmitter = tr1) -> AX transmitter != tr1) IN ctrl is true

AG(switch ^ tr1)-> AX (not tr1)

For all paths, if the current transmitter is tr1, and the control switches the transmitter, then the next transmitter is not going to be tr1. This was a general statement, so that the next transmitter was randomly selected.

-- specification (AG (switch = TRUE & transmitter = tr2) -> AX transmitter != tr2) IN ctrl is true

(Same as above)

-- specification (AG (switch = TRUE & transmitter = tr3) -> AX transmitter != tr3) IN ctrl is true

(Same as above)

-- specification AF cr.status = tr2 is true

AF(tr2 is on)

Finally for all paths, the car will be using transmitter 2. Introduced to make sure the car is switching the transmitter it is using.

-- specification AF (curr = name -> AX stat = TRUE) IN trans1 is true

If current transmitter is the same as the transmitters name, then that means the transmitter is on.

-- specification AF (curr = name -> AX stat = TRUE) IN trans2 is true

(same as above)

FAIRNESS ctrl.switch != cr.talk

Using only paths where switch and talk are not first initialized as being TRUE. This is so that they could start in alternate states, and then remain in alternating states (because the car was either talking, or switching)

FAIRNESS switch = TRUE & talk = FALSE

(Same as above)

FAIRNESS switch = FALSE & talk = TRUE

(Same as above)

FAIRNESS trans1.stat = TRUE & trans2.stat = FALSE & trans3.stat = FALSE

The initial state of the transmitters will not be such that they are all on or all off. Only paths where one is TRUE and the others are FALSE are selected.

FAIRNESS trans1.stat = FALSE & trans2.stat = TRUE & trans3.stat = FALSE

(Same as above)

FAIRNESS trans1.stat = FALSE & trans2.stat = FALSE & trans3.stat = TRUE

(Same as above)

Conclusions:

This was simply a simplification/abstraction of a real-world system. Using NuSMV to mathematically and logically describe a system helped with bug checking and finding if there were any logical issues with the system. Formally verifying the system helped to check the logical flaws in the code, leading to tweaks in the code. When generalizing to multiple cars and transmitters, I used several instances of processes of the modules. Although processes are deprecated for future builds, using them allowed me to easily add the number of cars/transmitters I had running. The specifications had to slightly be tweaked when scaling up the number of cars/transmitters, to account for them.