Modelling Reactive Systems (H) Assessed Exercise

Submission by
Sidhant Bhavnani
2482327B@student.gla.ac.uk

Answer 1

Refer to the attached file, frog-puzzle.pml.

N-Frog Implementation

My implementation is the extended version described in the question, with N frogs of each colour (and 2N+1 lily pads). The defined constants N and MAX can be changed to check this.
The implementation for checking the frog positions in the monitor do-loop is as follows:

- We first define a boolean variable and set it as true.
- Said boolean variable is updated with the value of the same boolean variable and the value of (state[i] == red) where i is in 0..N-1 and state is the array of pads. This loop checks if all the pads on the left-hand side have red frogs.
- We do the same for the right-hand side and check for yellow frogs by traversing states N+1..PADS-1 where PADS is the number of lily pads.
- If both the boolean cases are true and the middle pad (Nth pad) is null then we assign gameOver to be true.

Note to checker: Lines 13 to 21 can be commented and 22 to 24 can be uncommented to check the specific case of N=3.

Answer 2

Specification and Modelling Decisions

My model uses the following rendezvous channels to model communication between the processes: sensorsToSC, throttleToSC, brakeToSC, wheelToSC and SCToWheel.
The messages sent on the channels is as follows:

Channel	**Message Type**	**Messages**
sensorsToSC	mtype	on off
throttleToSC	mtype	faster **none**
brakeToSC	mtype	stop slower **none**
wheelToSC	int	x: x is an int
SCToWheel	mtype, int	faster,x: x is an int slower,x: x is an int stop,x: x is an int **cruise**, x: x is an int

Since I have used rendezvous channels the messages aren't stored within the channel and the program halts till the channel isn't polled. Thus, my model is completely synchronous and all messages that are sent must be read.
To accommodate for the synchronous nature, in my implementation I have altered the specification provided by adding a 'none' message on throttleToSC and breakToSC. This is to signify the states when the brake and throttle are not active, respectively.
An outcome of these extra messages and rendezvous channels is that my model is also able to capture what happens when breaks, throttle and sensors aren't active.
I have also included a 'cruise' message on SCToWheel to signify no change in speed. This is to model the cruise control of the car or the state where no message affecting the state of the model is received.

I use the following table to model what action to be taken with a combination of messages:

Break State	**Sensor State**	**Acceleration**	**wheel!**
none	on	faster	cruise
none	off	faster	faster
none	on	none	cruise
none	off	none	cruise
slower	on	faster	cruise
slower	off	faster	slower
slower	on	none	cruise
slower	off	none	slower
stop	on	faster	stop
stop	off	faster	stop
stop	on	none	stop
stop	off	none	stop

As we can see from this table, the order of priority is as follows: breaks > sensor > acceleration. With the exception of the sensor being on and a ‘slower’ message being sent by the breaks.

Property Verification

Claim: “If an on message is sent from the sensors process to the speedControl process, then the speed will not increase until a corresponding off message is sent”
To verify this claim I added a monitor process to my program and moved variables that were initially declared within speedContol, globally like they can be referenced by the monitor process as well.
The implementation is as follows:

- It checks if the sensor is on, if it is, it stores the currentSpeed in a sensorOnSpeed variable and moves to the label T1_monitor. If it is off, it repeats the check on the sensor being on by moving to the label in the beginning of the process, T0_monitor.
- Once the execution reaches the T1_monitor label, it checks if the sensor is still on, if it is then it asserts if the sensorOnSpeed

and the currentSpeed are the same.
If the assertion fails then the verification tool catches it.

- If the sensor is on and assertion is passed, it moves back to the T1_monitor label to repeat the check. If the sensor is off it moves back to T0_monitor to repeat the whole process.

Property Verification Result

The verifier did catch the counterexample of the brakes being used to stop the car while the sensors are on. In the trail attached we see this occurring close to step 350. Said monitor can be commented out to see the full execution without it halting.