Concurrent Programming Exercise 2

The aim of this Exercise is:

- to gain familiarity with the use of Uppaal system
- to gain practice in modelling and verification of simple concurrent programs

You should now be able to start Uppaal from the start menu in Windows. Save (download, don't open) the following files into your home library:

- PC_prob.xml
- Tie-breaker.xml
- Tie-breaker.q

You do not have to hand in any solution to this exercise.

Exercise 2.1

This exercise is to make sure that everything works and to give you an introduction to the Uppaal syntax and a practical demonstration of the different tools within Uppaal. Use Uppaal to open the project Tie-breaker.xml. The project implements a model of the tiebreaker algorithm for two processes. The code for the algorithm is:

```
var flag[2]:bool:=([2]false)
var turnt:int:=1
process M1::
   do true ->
      flag[1] := true
      turn := 1
      do flag[2] and turn = 1 ->
         skip
      od
      # critical section
      flag[1] := false
      # non-critical section
   od
end
process M2::
   do true ->
      flag[2]:= true
      turn := 2
      do flag[1] and turn = 2 ->
         skip
      od
      # critical section
      flag[2]:= false
      # non-critical section
   od
end
```

Since the protocol is symmetric for the two processes, they are modeled by the same template, mutexP. As you can see, the template has four states. The initial state is marked with a ring. When the process wants to enter its critical section, it takes a transition to next state and thereafter to a waiting state. Here it is blocked until the blocking condition is falsified. The fourth state is the critical section.

There is also a second template, Hurry. The only purpose with this template is to force the execution forward. Otherwise, the processes might choose to stay forever in the same state.

If you look at the process assignments, you can see that two processes are assigned the mutexP template. Note also the assignments of the flags. The processes are included in the system definition. Before running the simulator or verifier, you can check the syntax from the menu.

In the simulator, you may run an execution of the two processes either by random or stepwise. For each step you can select among possible orders. Running a simulation can be viewed as running a test execution. This means that one out of all potential orders is executed.

In the verifier, it is possible to question the system. The verifier examines all states and orders until it can generate an answer. A set of queries is already given for this exercise, Tie-breaker.q. Use load queries to include it in your system. Now you can run each query by selecting it and press model check.

It is possible to verify different properties by asking the system for global invariants, reachability, etc. The syntax is as follows:

- A[] φ means that φ is true in all states (i.e., φ is a global invariant)
- E<> φ means that it is possible to reach a state where φ is true (i.e., φ is reachable)
- A<> φ means that φ will eventually be true regardless of execution trace
- E[] φ means that it is possible to run an execution where φ is true in every state
- φ --> ψ means that from all states where φ is true, ψ will eventually be true (liveness)

Exercise 2.2

In this exercise, you will use the tools in Uppaal to verify your own solution. Recall the producer consumer problem that you solved in Exercise 1. Now you are supposed to verify your solution to that problem. Use the file PC_prob.xml and extend the model with your solution. When you have finished the model and checked the syntax you should define a set of queries to verify safety and liveness. Run the queries. Is your solution correct and does it have safety and liveness?

Note: The verification is no better than your model and your queries:

- If the model doesn't implement the same behavior as the implementation, your proof is not valid
- If your queries are not well formulated, the results are not valid
- If your set of queries is not well chosen, the behavior of the model is not well covered (proven)