BCS2213 – Formal methods

Teaching assignment 8. Learning UPPAAL. Modelling systems using timed automata.

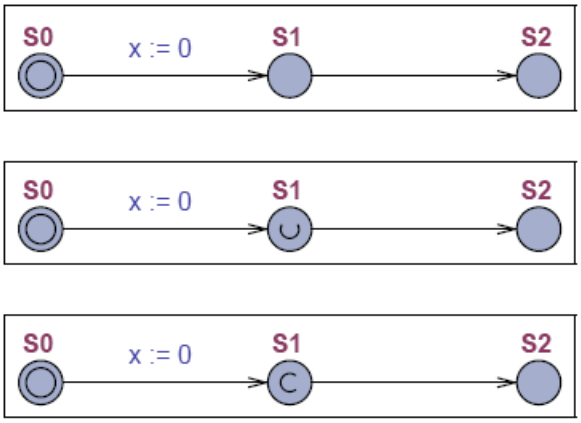
1. Run UPPAAL Toolbox.

2. Learning the different types of locations.

There are three different types of locations in UPPAAL:

- Normal locations
- Urgent locations
- Committed locations.

Create new model and draw the automata depicted on Figure below. Name processes P0, P1 and P2 correspondingly. Define the clocks x locally for each automaton (for it open the sub-tree of their templates in the Project tree; there you will see a Declarations label under the template). Click on it and define clock x. Repeat for the other two automata.



The location marked "U" is *urgent* and the one marked "C" is *committed*. Try them in the simulator and notice that when in the committed state, the only possible transition is always the one going out of the committed state. The committed state has to be left immediately.

To see the difference between *normal* and *urgent* state, go to the verifier and try the properties:
E<> P0.S1 and P0.x>0 (it is possible to wait in S1 of P0).
A[] P1.S1 imply P1.x==0 (it is not possible to wait in S1 of P1).

Time may not pass in an urgent state, but interleaving with normal states are allowed as you can see in the simulator. Thus, urgent locations are "less strict" variants than committed ones.

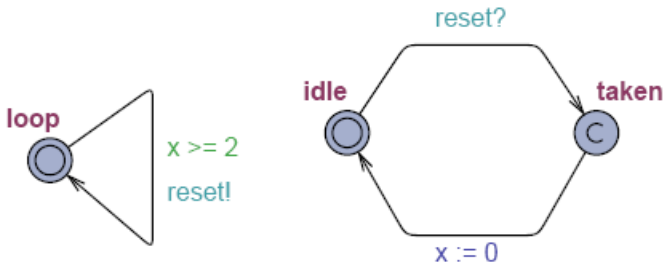
Make screenshot from this UPPAAL model and put them into Word file with possible comments. Save the file with the name lab_8_<your_ID>.doc

3. Modelling systems using timed automata

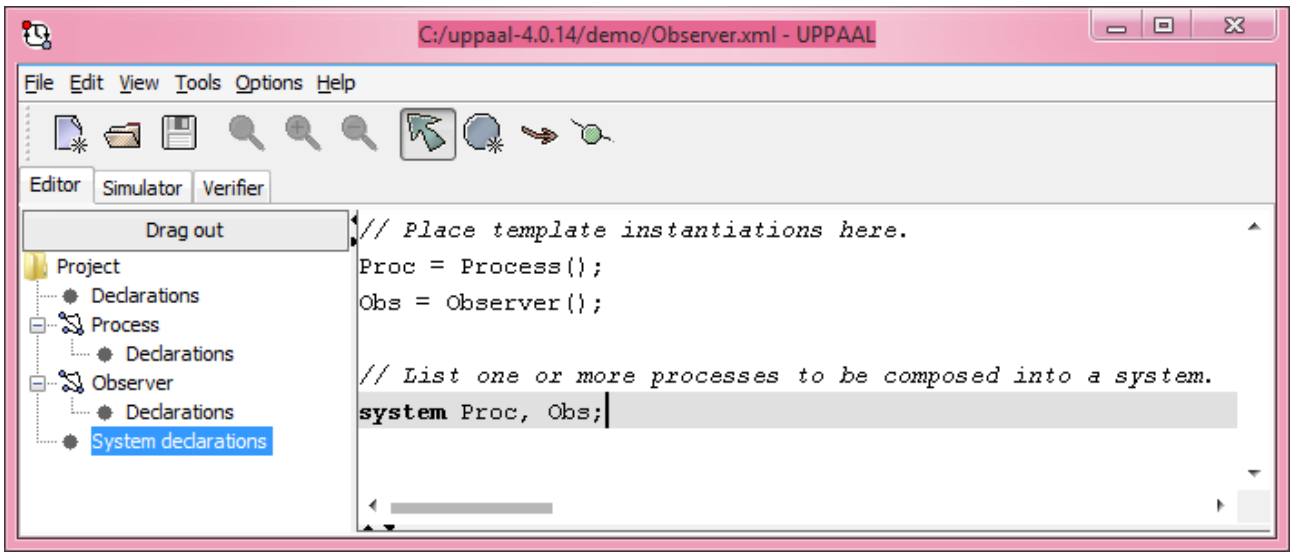
The clocks are the way to handle time in UPPAAL. Time is continuous and the clocks measure the time progress. It is allowed to test the value of a clock or to reset it. Time will progress globally at the same pace for the whole system.

To grasp how the time is handled in UPPAAL we will study a simple example of *Observer*. Normally, an observer is an automaton used for detecting events without perturbing the observed system. In our case the reset of the clock (x = 0) is delegated to the observer to make it work.

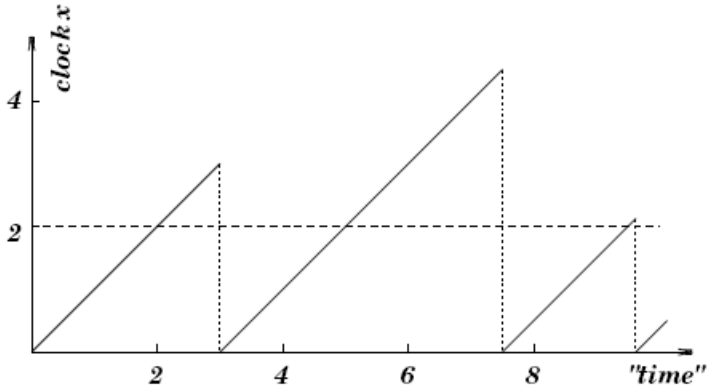
Figure below shows the first model with its observer. Time is used through *clocks*. In the example, x is a clock declared as clock x; in the *global* (Project) Declarations section. A channel reset is used for synchronization with the observer, which is also declared (as chan reset;) in the Declarations section. The channel synchronization is a hand-shaking between reset! and reset? In this example the clock *may be* reset after 2 time units. The observer detects this and actually performs the reset.



Draw the model, name the automata (templates) Process and Observer, and define them in the system. A new template is created with *Insert Template* in the *Edit* menu. Notice that the state taken of the observer is of type *committed*.



To interpret what you see we will use queries and modify the system progressively. The expected behaviour of our system is depicted on next Figure.

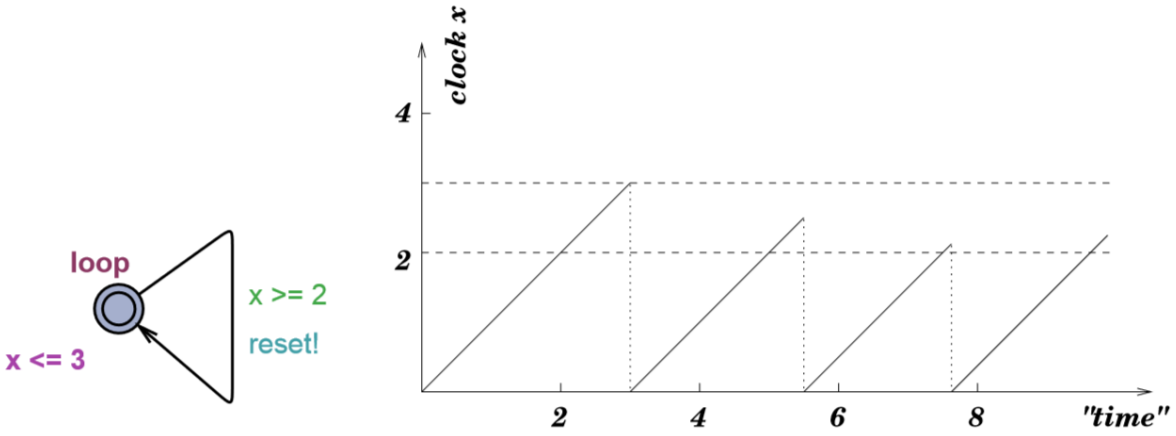


Try these properties to exhibit this behaviour:

A[] Obs.taken imply x>=2 : all fall-down of the clock value (see curve) are above 2. This query means: for all states, being in the location Obs.taken implies that x>=2.

E<> Obs.idle and x>3 : this is for the waiting period, you can try values like 30000 and you will get the same result. This question means: is it possible to reach a state where Obs is in the location idle and x>3.

Add now an invariant to the loop location as shown on next Figure.



The invariant is a progress condition: the system is not allowed to stay in the state more than 3 time units, so the transition has to be taken and the clock reset in our example.

To see the difference, try the properties:

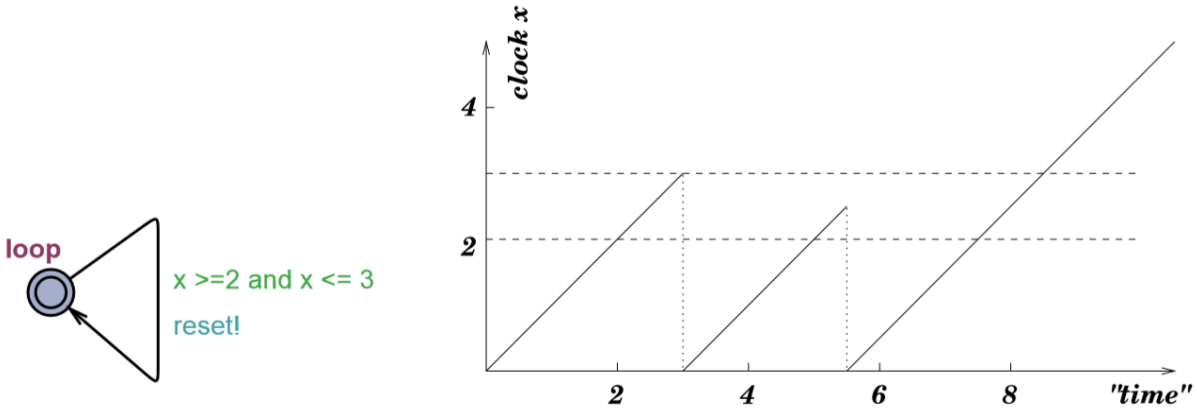
• A[] Obs.taken imply (x>=2 and x<=3) to show that the transition is taken when x is in the interval [2,3].

• E<> Obs.idle and x>2 : it is possible to take the transition with x in the interval (2,3].

• A[] Obs.idle imply x<=3 : to show that the upper bound is respected.

The former property E<> Obs.idle and x>3 no longer holds.

Now, remove the invariant and change the guard to x >= 2 and x <= 3. You may think that it is the same as before – but it is not! The system **has no progress condition** anymore, just a new condition on the *guard* now. Figure below shows the new system.



As you can see the system may take the same transitions as before, but there is now a deadlock: the system may be stuck if it does not take the transition after 3 time units. Retry the same properties, the last one does not hold now. Actually you can see the deadlock with the following property: A[] x>3 imply not Obs.taken, that is after 3 time units the transition can not be taken any more.

Make screenshots from your second model and add them into your lab_8_<your_ID>.doc file.

4. If you still have time and want to have additional marks develop the model of 4 Vikings, was considered at the lecture 11. Change the model in order it has 5 Vikings. Is it still possible to cross the bridge in 60 min? If so, what should be the time to cross the bridge by this 5 Viking?

5. Upload lab_8_<your_ID>.doc into Kalam for evaluation.