BCS2213 – Formal methods

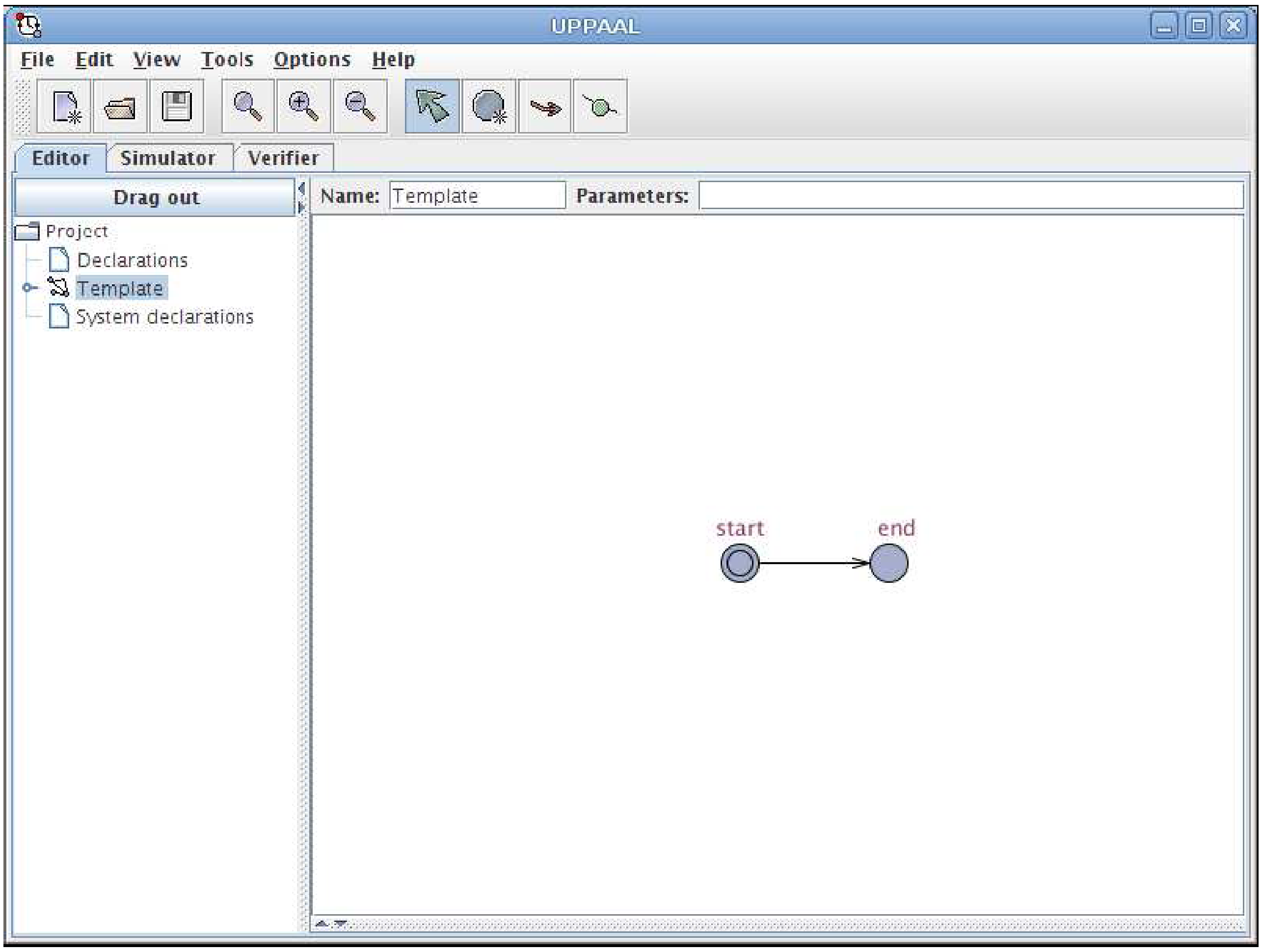
Teaching assignment 6. Learning UPPAAL. Modelling systems using timed automata.

1. Run UPPAAL Toolbox.

UPPAAL is a Toolbox for validation (via graphical simulation) and verification (via automatic model-checking) of real-time systems. It consists of two main parts: a graphical user interface and a model-checker engine.

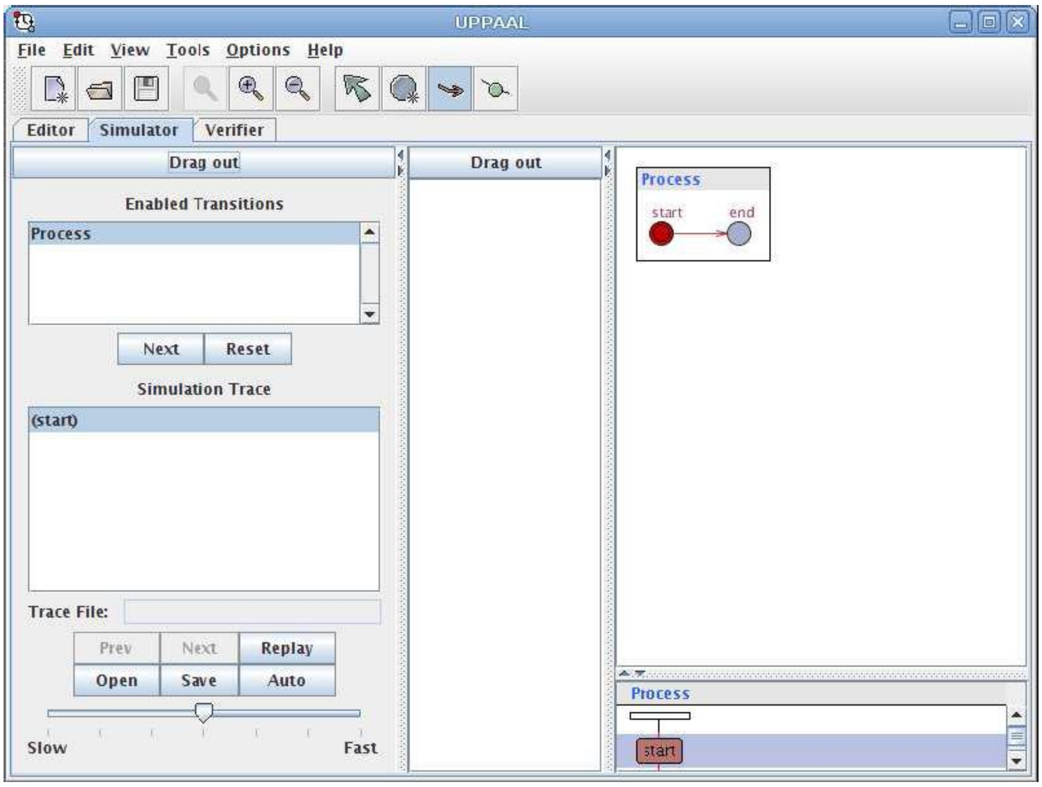
The UPPAAL main window has two main parts: the menu and the tabs - the *editor*, the *simulator* and the *verifier*.

When you start Uppaal, there is one location pre-created in the drawing area. It is the *initial* location of the automaton, so it has an additional circle inside. To add a second location, click on the "Add Location" button in the tool bar (tool tips will help you) and then click in the drawing area, a bit next to the initial location. Use the "Selection Tool" to give them the names start and end. Then choose the Add Edge button, click on the start location and on the end location (see figure below).



2. Simulator mode

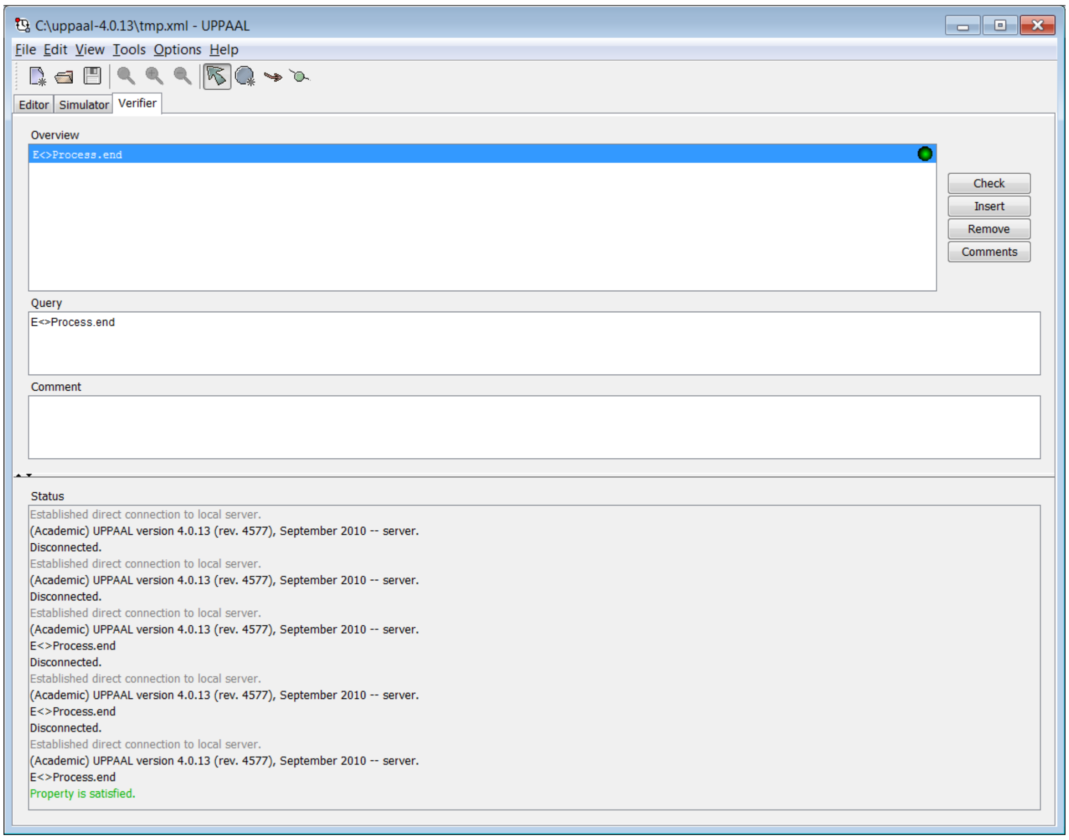
Now, click on the Simulator tab to start the simulator, click on the yes button that will pop up and you are ready to run your first system.
Figure below shows the simulator view. On the left you will find the control part where you can choose the *transitions* (upper part) and work on an existing trace (lower part). In the middle are the variables and on the right you see the system itself. Below the system, you will see what happens in which process.
To simulate our trivial system pick one of the *enabled transitions* in the list in the upper left part of the screen. There is only one transition in our example. Click Next. The process view to the right will change (the red dot indicating the current location will move) and the simulation trace will grow. You will note that more transitions are actually not possible, the system is deadlocked.



3. Verification mode

Click on the Verifier tab (see figure below). The upper section allows you to specify *queries* to the system. The lower part logs the communication with the model-checking engine.

Enter the text E<>Process.end in the *Query* field below the Overview. This is the Uppaal notation for the temporal logic formula ∃ ◊Process.end and should be understood as "it is possible to reach the location end in automaton Process". Click Check to let the engine verify this. The bullet in the overview will turn green indicating that he property indeed is satisfied.



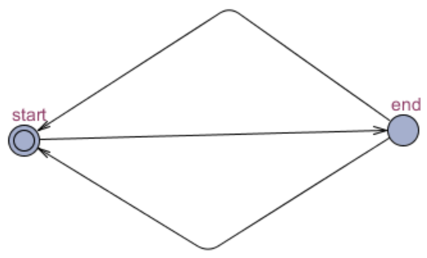
Overview: E<>Process.end

Query: E<>Process.end

```
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
E<>Process.end
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
E<>Process.end
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
E<>Process.end
Property is satisfied.
```

4. Modelling more complex transitions.

Modify your example in order it has two possible transitions from end to start node (see figure below).



Declare Boolean variable x (bool x; in Project Declarations), and allow the first state transition if x is true, and second – if x is false (so you have guards, correspondingly, x==true, x==false). Update of the state transition will be correspondent assignments (x=false, x=true).

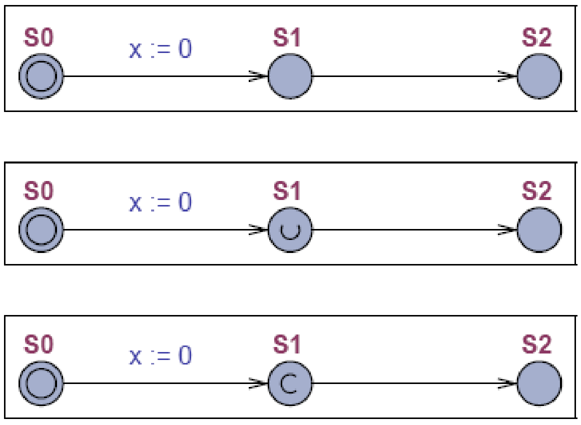
Track your model in simulation mode.

Save your first UPPAAL labsheet with the name containing your ID and (UPPAAL system has xml extension) and upload it into Moodle.

5. Learning different types of locations.

There are three different types of locations in Uppaal:
Normal locations with or without invariants (like x == true above),
Urgent locations and
Committed locations.

Create new model and draw the automata depicted on Figure below and name them P0, P1 and P2. 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 in question. 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.

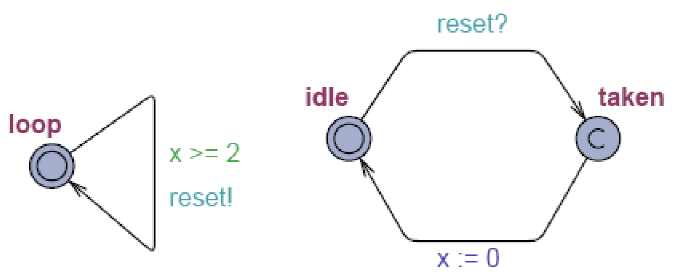
Time may not pass in an urgent state, but interleavings with normal states are allowed as you can see in the simulator. Thus, urgent locations are "less strict" variants than committed ones.

6. Modelling systems using timed automata

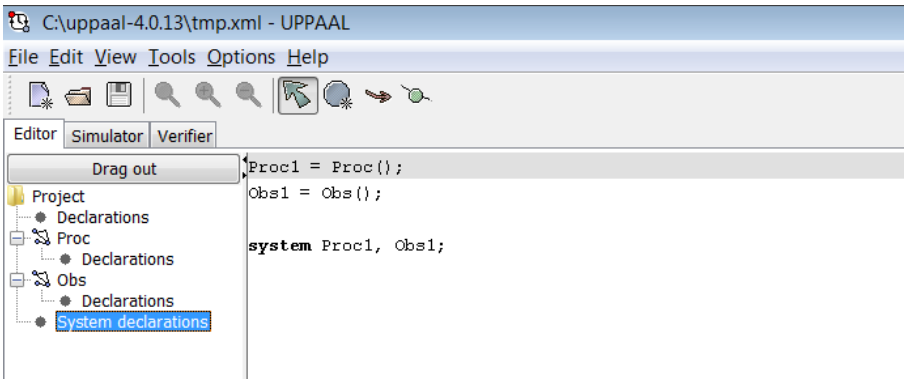
The clocks are the way to handle time in Uppaal. Time is continuous and the clocks measure time progress. It is allowed to test the value of a clock or to reset it. Time will progress globally at the same pace for the whole system.

To grasp how the time is handled in Uppaal we will study a simple example of *Observer*. Normally, an observer is an add-on automaton in charge of 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. Note that by doing this, the original behaviour of the system – with the clock reset directly on the transition loop to itself – is not changed.

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? So in this example the clock may be reset after 2 time units. The observer detects this and actually performs the reset.



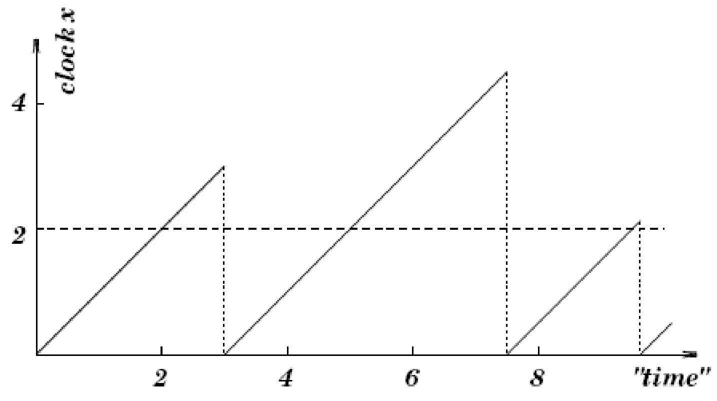
Draw the model, name the automata (templates) Proc and Obs, define them in the system. A new template is created with *Insert Template* in the *Edit* menu. Notice that the state taken of the observer is of type committed.


C:\uppaal-4.0.13\tmp.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Project
Declarations
Proc
Declarations
Obs
Declarations
System declarations

```
Proc1 = Proc();
Obs1 = Obs();

system Proc1, Obs1;
```

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[] Obs1.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<> Obs1.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.

11. Save your second UPPAAL labsheet with the name containing your ID and (UPPAAL system has xml extension) and upload it into Moodle.