Lecture

Topics:

- Introduction to real-time systems, motivation and validation techniques
- Introduction to Uppaal: modelling, simulation and verification of timed automata

Readings

1. Read Chapter 1 of [Shaw] for a general view of real-time systems.
2. Read Chapter 1 of [LO] (Laplante & Ovaska) for a another view of real-time systems.
3. Skim Chapter 1 (pages 1–18) in [BK] (Baier & Katoen) for a motivation for the formal approach to real-time validation adopted in this course.
4. Read sections 1-3.2 in [Upppal-ST] (Small Tutorial) for a gentle introduction to the Uppaal tool.

All of these texts are to be found in the Week 01 module on DTU Learn.

Exercises

The main purpose of these exercises is that you get acquainted with the Uppaal system, in particular the editor and the simulator. Today's focus is on untimed models.

Before the class, you should install 4.1.26 of the Uppaal tool found at `http://uppaal.org`

Exercise 1: Simple Light Control

Start Uppaal and open the file for the simple light control system presented at the lecture (`SimpleLight.xml` from the Week 01 module on DTU Learn) and experiment with the system, in particular, with the simulator.

Make a more refined light control system, where the light can be off, dimmed or bright. The On button should be used to toggle between the two light modes (dimmed and bright). You may decide yourself which light mode is entered initially. Model the system in Uppaal and experiment using the simulator.

Enhance the light control system to return to the last light mode used (dimmed or bright) when being switched on again.

Now, enable fine-grained dimming of the light control such that there are five levels of dimmed light between off and bright. Use a local state variable to represent the degree of dimmedness. You may add extra buttons for dimming control, if you like.

Exercise 2: Goat/Wolf/Cabbage Puzzle

A man is travelling with a goat, a wolf and a cabbage. They reach one shore of a river which they must cross using a little boat.

Unfortunately, the boat can carry only the man and *one* of the others. Furthermore, if the goat is left unattended with the cabbage, the goat will eat the cabbage, and if the wolf is left unattended with the goat, the wolf will eat the goat.

You are now going to make some Uppaal models of the system:

1. First make a model which allows for all scenarios of transportation — whether unsafe or not. Simply model what is physically possible whether sensible or not.

   The model should have an automaton for each of the man, the wolf, the goat and the cabbage which clearly shows the position of the item. Initially, they should all be on the same shore. Use synchronization on dedicated event channels to model the man taking a particular item from one shore to another.

   Try to use the simulator to device a safe way (where nobody is eaten) of crossing the river. Here, you are controlling the will of the man by selecting among the possible moves.

2. In the Uppaal Verifier tab, formulate an *invariant* as a Uppaal query of the form:

   `A[]` ϕ

   where ϕ is a *state predicate* you write expressing that nobody is in the risk of being eaten. This query asks whether all execution paths (`A`) will satisfy this at all (`[]`) points of time. You may use $P.L$ to express that process P is currently at location L and combine such with the usual boolean operators.

   Use the Uppaal verifier to check the invariant (which will fail, of course). In the Options menu enable diagnostic trace (shortest) and after the verification, run the diagnostic trace in the simulator to see how the invariant may fail to hold.

3. Now give the man some *common sense*. This means that he should never leave somebody behind to be eaten. For this, the man must be equipped with some perception of the current state. Since in Uppaal, it is not possible for one process to refer to the inner workings of other processes, the man must build his own idea of where the items are, stored in some local state variables. (You might imagine the man to be blind.) Using his memory, he may now make safe (but not necessarily clever) decisions.

   The query from above should now be satisfied. Is it?

   Once done, you may try out a query of the form:

   `E<>` ψ

   where ψ is a state predicate you write expressing that all have (safely) arrived on the oppostite shore. This query asks whether there exists an execution path(`E`) where eventually (`<>`) ψ holds. If so, you can run it in the simulator to see a solution to the puzzle.

4. (Optional) The man may still make a number of useless sailing trips. Try to equip the man with some "intelligence" in the form of a plan for the journey, possibly using a "stage" variable. With this, you should be able to verify

   $$\texttt{A<>}\ \psi$$

   which states that *all execution paths* will now lead to the desired state.

   When working with untimed models, for the verification to work as expected you must force the system to take transitions by marking the states as **urgent** (using right-click). In this case it is only necessary for the Man, who is the driving force in the system. The technical reasons behind this will be explained later in the course.