Prof. Dr. A. Podelski, Dr. B. Westphal
L. Musliu

Wintersemester 2017/18

Real-Time Systems

http://swt.informatik.uni-freiburg.de/teaching/WS2017-18/rtsys

Exercise Sheet 7

Early submission: Monday, 2018-01-29, 14:00 Regular submission: Tuesday, 2018-01-30, 14:00

Exercise 1 — Queries (2/20 Points)

Task 3.(ii).f) of Exercise Sheet 6 asked for a query to check that "a sensor failure is detected 10 s after the failure the latest". Consider the corresponding model with query which is available with the ILIAS exercise.

(i) First, explain the query and outcome of the check in your own words (you may re-use your own submission to Task 6:3.(ii).f) if you like).

Then disable (or delete) the two edges to `fail` which originate at `send` and at the unnamed, rightmost location in the sensor and re-check the query.

What do you observe? Discuss! (2)

Exercise 2 — Disjunctive Constraints (1/20 Points)

Figure 1: Disjunctive clock constraint.

Assume we need to model timed behaviour where a transition from a location l to a location m may happen at any time except for the interval $[3, 4]$ after system startup. A naïve solution as shown in Figure 1 neither satisfies the definitions from the lectures nor is it accepted by Uppaal.

(i) Is it principally impossible to model such behaviour, or is there a way to obtain the desired behaviour with a well-formed Uppaal model? If no, argue why not, if yes, explain your approach. (1)

Exercise 3 — TA vs. DC Implementables (5/20 Points)

In Exercise 3.(ii) from Exercise Sheet 4, the task was to describe a design for a jamming device using implementables. One implementable from the tutorial read

$$F :\equiv \lceil o \rceil \xrightarrow{1} \lceil \neg o \rceil$$

where 'o' modelled "all four channels free".

(i) Check (using Uppaal) whether your timed automata model of the jamming device correctly realises implementable F. (3)

(ii) To cross-check your solution for Task (i), modify your timed automata model of the jamming device such that it *does not* realise F any more and re-check the resulting model. (1)

(iii) Check the query '`A<>` φ' (where φ characterises reaching the "bad location" of the constructed observer) for the changed model from Task (ii) and interpret the outcome. (1)

Exercise 4 — Reachable Zone Graph (3/20 Points)

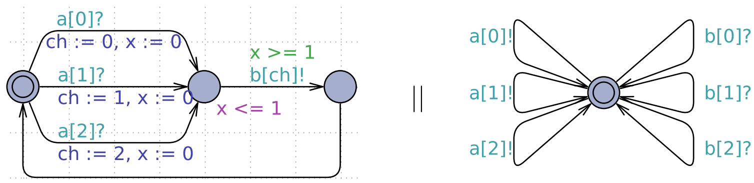


Figure 2: Network of timed automata with a timed response behaviour.

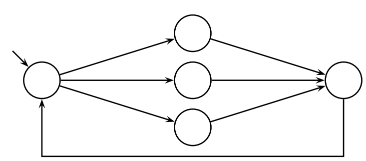
```
% verifyta -u -s AB.xml
Options for the verification:
  Generating no trace
  Search order is breadth first
  Using conservative space optimisation
  Seed is 1515780592
  State space representation uses minimal constraint systems

Verifying formula 1: A[] true

 -- Formula is satisfied.
 -- States stored : 9 states
 -- States explored : 9 states
 -- CPU user time used : 0 ms
 -- Virtual memory used : 24168 KiB
 -- Resident memory used : 4744 KiB
```

Figure 3: Statistics from model-checking the network from Figure 2.

Consider the network of timed automata shown in Figure 2. The automaton on the left accepts inputs on the channels $a[0]$, $a[1]$, and $a[3]$, and responds to each input on $a[i]$ with an output on the matching channel $b[i]$ one time unit later. To this end, the number of the input channel is stored in the variable `ch` to be used in the subsequent output.
At first sight, one would expect the reachable zone graph of the network to look like this:



That is, one initial configuration, one configuration for between input on $a[i]$ and reply $b[i]$, and one before going back to the initial configuration.

Model-checking the network against query `A[] true`[1] with the Uppaal command-line model-checker yields the statistics shown in Figure 3.

(i) Why do we see a number of 9 states (i.e. configurations) explored, instead of the 5 we expected? (2)

(ii) Can we change the model such that the behaviour on a and b (including timing) is preserved but only the expected 5 configurations are explored? (1)

By-the-way: the model from Figure 2 can be written more concisely using select *statements:*

Here, `c : chID` *works like a free local variable with the edge as scope; the edge is enabled if one can choose (or select) a value for* `c` *such that the guard is satisfied etc.*

[1] We check an invariant which is satisfied by the model to get an idea of the number of reachable configurations, since for a satisfied invariant, the whole set of reachable configurations needs to be examined. Otherwise, the search could stop as soon as a counter-example is found. Yet, the particular trivial query '`A[] true`' has to be used with care: some tools may recognise its triviality and immediately report "yes, satisfied" without doing any examination. Uppaal seems not to be a tool of this kind.

Exercise 5 — Deadlock (4/20 Points)

The query language actually supports one more atom in addition to locations and constraints, namely

`deadlock`

so that, e.g., `A[] deadlock` is a valid query.

(i) What is the formal semantics of "`deadlock`"? Explain the intuition in your own words. (2)

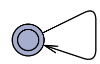
Hint: there are online documentation and offline tutorials and manuals available for Uppaal.

(ii) Why would anybody be interested in using this atom?

Provide a timed automata model which is supposed not to have a deadlock in this sense, check that it doesn't have a deadlock.

Then introduce an error into your model to obtain a new model which has a deadlock and check that you introduced the error correctly. (1)

(iii) If the timed automaton



is added to your defective model from the previous task, does the resulting network still have a deadlock?

Answer the question without actually using the tool. (1)

Exercise 6 — Model-Checking Cost (5/20 Points)

Recall the WFAS model from Exercise Sheet 6. The model of the deterministic jamming device is parameterised in upper and lower bound on reaction time (`eps1` and `eps1`).

(i) Collect statistics on the explored/stored states when checking your queries from Exercise Sheet 6 using the command line model-checking tool `verifyta`(1).

Consider at least the value pairs $(1,5)$, $(0,0)$, and a third one of your choice. (3)

Hint: research results should be reproducible, so please provide a small shell script (or makefile (or ...)) so that your tutor can easily check your claims from the submission.

(ii) Which queries are the most and second-most expensive ones wrt. explored/stored states? (1)

(iii) What is the reason for the different costs of checking the model with different value pairs? (1)