Real-Time Systems

http://swt.informatik.uni-freiburg.de/teaching/SS2014/rtsys

Exercise Sheet 6

Early submission: Monday, 2014-07-30, 12:00 Regular submission: Tuesday, 2014-07-31, 10:00

Exercise 1: Extended Timed Automata (5/20 Points)

With Extended Timed Automata, we introduced committed locations.

- Explain in your own words, possibly using examples (different from the one in the lecture) the difference between urgent and committed locations. (3)
- We explained urgent locations by a syntactical transformation, urgent locations are thus not part of a Extended Timed Automaton tuple. Committed locations are. Could we also explain committed locations by a syntactical transformation? In other words: do committed locations add expressive power to Pure Timed Automata? (2)

Exercise 2: Model-Checking with Uppaal (5/20 Points)

Consider the Off/Light/Bright model from Exercise Sheet 4.

(i) Use the model checker to verify whether the original user can reach the Bright location. (1/5)

(ii) Use the model checker to verify that your modified user from Sheet 4, Exercise 2, part (iii) cannot reach the Bright location as requested. (1/5)

(iii) Check whether the original user is able to keep the lamp at location Bright for more than 5 time units. (1.5/5)

(iv) Check whether the original user is able to switch the lamp to Bright twice. (1.5/5)

Explain your approach.

Exercise 3: DC and Uppaal (5/20 Points)

Consider Exercise 2 of Exercise Sheet 2 (requirements for traffic lights).
Which of those requirements is testable, which one is not? If yes, give a test automaton (observer, monitor) similar to the one we've used in the proof sketch for Theorem 6.4, if not, explain why not. (5)

Exercise 4: Scalability (5/20 Points)

Consider the model of Fischer's protocol in Uppaal's `demo/` directory.

- Explain in your own words how the protocol achieves mutual exclusion. (1)

  Hint: your explanation will be much more readable if you include the automaton as a figure.

- Fischer's protocol is parameterised in the number of processes. The number of processes to be considered can easily be chosen in the global declarations. Measure the time and memory it takes Uppaal to verify the absence of deadlocks in instances with 2, 3, 4, ... processes, provide your data in a table, and present them graphically in an appropriate plot.

  Admit at most 1h of verification time for each task – what is the largest instance you can check within this time limit?

  (2)

  Hint: The command-line tool `verifyta`(1) from the Uppaal package may be better suited for this kind of benchmarking task. Under Unix, `ulimit`(1) or `limit`(1) can be used to enforce time limits, `time`(1) can be used to measure time and memory consumption. To make your experiment (more) repeatable, give the characteristics of your experimental setup (Uppaal version, CPU, Memory size, Operating system).

- Repeat the experiments of the previous task with the query

  $$\texttt{E <> forall(i : id\_t)P(i).wait}$$

  for the same instances for which absence of deadlocks could be verified in 1h and the number of instances for which verification first did not succeed within 1h.

  Measure both, breadth- and depth-first search (cf. `verifyta --help`), provide your data in a table, and present them graphically.

  Compare the figures from the previous task and the ones from this task. Do you have a hypothesis how to explain the outcome?

  (2)