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

Summer term 2019

**Softwaretechnik/Software Engineering**

`http://swt.informatik.uni-freiburg.de/teaching/SS2019/swtvl`

# Exercise Sheet 5

Early submission: Wednesday, 2019-07-10, 12:00   Regular submission: Thursday, 2019-07-11, 12:00

## Exercise 1 – Computation Graph / Transition Graph (8/20 Points)

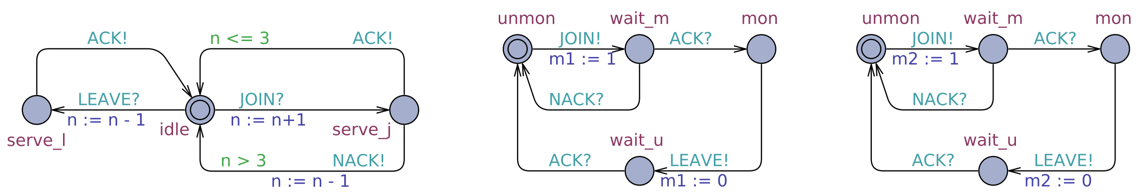

Figure 1: Network of Communicating Finite Automata

Provide the reachable part of the transition graph of the CFA model shown in Figure 1.
*Hint: Make sure to clearly indicate the initial configuration(s).*

## Exercise 2 – Basic Behavioural Model Analysis (7/20 Points)

The file `sensormaster.xml` provides a model of the sensor/master system from Exercise Sheet 4. Recall the requirement from Exercise Sheet 4, that masters only need to monitor a bounded number of sensors.

(i) Use the Uppaal[1] simulator to create one computation path which shows that the model is able to exhibit the simple scenario that one sensor joins a master and then leaves again. (1)

*Hint: Include a screenshot showing the final state of the computation path and the sequence diagram into your submitted document; and submit the `.xtr` file of the simulation as well so that your tutor can reproduce your result.*

(ii) a) Use the verifier to check whether, for some master, it is possible to monitor the allowed maximum number of sensors (as given by global constant/parameter `max_sns`).
What is the result? How should this result be interpreted? (2)
*Hint: That is, what can we conclude from the result of the above check about the modelled design idea?*

b) Use the verifier to check whether it holds that for each master, the value of `n` never exceeds the assumed maximum number of sensors. Some queries have been proposed for the check and are included in `sensormaster.xml`.
What are the results? How should these results be interpreted? (2)

*Hint: Submit a copy of `sensormaster.xml` with your 1.(ii).a) query filled in; make sure that your tutor understands which of the many files you submit is to be considered for this task.*

[1]See Appendix A for instructions.

(iii) The author of the model has expressed doubts about the model's correctness. One invariant that needs to hold for the model to be correct (according to the requirements department) is that if the value of variable `m` in any sensor is a proper master's id, then the value of `n` of this master must not be 0.

Explain how the outcome of checking the corresponding query in `sensormaster.xml` (unfortunately) confirms these doubts. (1)

(iv) Explain the reason for the incorrectness of the model according to Task (iii) and suggest an as small-as-possible change that preserves the outcomes of the Tasks (i) and (ii), but changes the outcome of the previous check. (1)

*Hint: Also submit the changed model in a separate file.*

*Hint: If you do not spot any reasons for the incorrectness, ask your tutor for clues (together with a clear description of the state of your investigation and your hypotheses so far).*

## Exercise 3 – Advanced Model Analysis (3/20 Points + 5 Bonus)

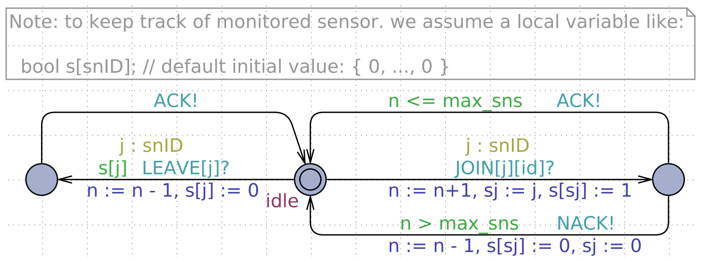
Note: to keep track of monitored sensor. we assume a local variable like:
bool s[snID]; // default initial value: { 0, ..., 0 }
ACK!
j : snID
s[j]
LEAVE[j]?
n := n - 1, s[j] := 0
idle
n <= max_sns
ACK!
j : snID
JOIN[j][id]?
n := n+1, sj := j, s[sj] := 1
n > max_sns
NACK!
n := n - 1, s[sj] := 0, sj := 0

Figure 2: Sketch of design idea.

After *counting* the number of sensors in the masters has been solved, the model should be extended such that the masters keep track of *which* sensors they are monitoring.
A proposed solution is sketched in Figure 2.

(i) Extend the model from Task (iv) of the previous exercise such that the extended model implements the sketch[2] given by Figure 2. (1)

*Hint: Submit the resulting file; make sure that the model includes comments that point out where you needed to 'fill in' things missing in the sketch (and that these comments are clearly recognisable by your tutor).*

(ii) There are two important correctness properties of the sensor/master system, for which the requirements department came up with the following (untypically precise) phrases:

a) For each master $j$ and sensor $i$, it holds that, if the $i$-th entry of the value of $j$'s `s` variable is not false (or: not 0), and if $j$ is idle, this implies that the value of `m` in $i$ is $j$.

b) For each sensor $i$ and master $j$, it holds that, if $i$ is monitored and its `m` has value $j$, this implies that the $i$-th entry of $j$'s `s` value is not false.

Formalise these requirements in UPPAAL query language. (1)

[2]Here, 'sketch' is used in the sense that the design department has just quickly drawn Figure 2 and (for good reason) assumes that you, as an (at least half) CFA-expert, will know how it defines a well-formed UPPAAL model. That is, only changing the CFA of template 'Master' may not be sufficient: The declaration of `s` needs to be introduced at the right place, and other smaller issues like this one may need to be resolved.

(iii) Check your queries from Tasks (ii) on your model from Task (i), once with 1 master (as in the given model), and once with 2 masters.[3]

What are the results? How should these results be interpreted? (1)

(iv) Some experienced designer had early doubts on the correctness of the proposed solution, yet the team did not want to rely on 'feelings' but wanted to be convinced and insisted on checking the model.

Your results from the previous task should prove the experienced designer right.[4]

Provide a comprehensive[5] description of what goes wrong, both technically and intuitively. Outline a solution (and model and check it, if you like). (5 Bonus)

## Exercise 4 – Computational Costs of CFA Verification (2/20 Points)

In this exercise, the task is to collet measures on the verification procedures as implemented in the UPPAAL tool on some particular configurations of `sensormaster.xml` (i.e. the one you downloaded).
Measure used time, virtual memory, and the number of explored states for a configuration with

`max_sns` = 3, `num_ms` = 3, `num_sn` = 5

for both queries from 2.(ii).b) again.

(2)

*Hint: The point of this exercise is to (at least once) practice empirical software engineering research (cf. IEEE 610.12, entry 'Software Engineering', meaning (2) (in Lecture 1)).*

*That is, we need to know what we are measuring (what does 'CPU user time' mean? (the man-page `time(1)` may serve as a start!)). Then, we need to decide how many executions of the same checking task to measure (why is one execution in general not enough?). Furthermore, the results need to be reproducible (you may want to use the command line verifier (cf. Appendix A) and scripting). Also include the specifications of the computer on which you performed your measurements, and state by which procedure you obtained your timing measurements (or just submit your script, too).*

*Hint: If you like (yet no points awarded here), you may state a hypothesis on the reasons for the observations, and also conduct further measurements to state a hypothesis on the (theoretical) complexity of CFA model-checking (in, e.g., number of sensors).*

[3]Note that, in the models that we use here, the global constants/parameters `num_ms` and `num_sn` determine the number of masters and sensors to be considered for simulation and verification.

[4]If this is not the case, immediately contact your tutor. `;-)`

[5]Comprehensive in the sense that you have very high confidence that at least 90 % of your fellow students who did not work on this task would be convinced of your analysis and consider the analysed design idea proven incorrect. In other words: as high-quality you would write if this task were your assignment of the day in an industry job; given to you by the boss of the experienced designer, since the 'big boss' wants to get an unbiased analysis.

## A Command Line Usage Instructions

Uppaal and a command line verifier are available on the Linux machines in the computer pool. To run Uppaal, use the following command:

```
/usr/local/ufrb/uppaal/uppaal-4.1.19/uppaal
```

To run the Uppaal command line verifier, use the following command:

```
/usr/local/ufrb/uppaal/uppaal-4.1.19/bin-Linux/verifyta -u <model> <query>
```

where `<model>` is the file where your model is stored and `<query>` is the name of a text file containing the query to verify (this is just plain text, can be exported from Uppaal). The output of running `verifyta` looks like the following:

```
Options for the verification:
  Generating no trace
  Search order is breadth first
  Using conservative space optimisation
  Seed is 1467050321
  State space representation uses minimal constraint systems

Verifying formula 1 at line 1
 -- Formula is satisfied.
 -- States stored : 145 states
 -- States explored : 109 states
 -- CPU user time used : 430 ms
 -- Virtual memory used : 25288 KiB
 -- Resident memory used : 4908 KiB
```

Note: if you *have access to other versions of* Uppaal then the one named above *and want to use one of those*, please make sure that the version named above is able to read the model and trace files, since this is the version that the tutors will use. The (otherwise remarkably) stable Uppaal tool faced some regressions regarding file formats lately (4.1.22, for example, is know to be okay).