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

Sommersemester 2017

Softwaretechnik/Software Engineering

`http://swt.informatik.uni-freiburg.de/teaching/SS2017/swtvl`

Exercise Sheet 5

Early submission: Wednesday, 2017-07-12, 12:00 Regular submission: Thursday, 2017-07-13, 12:00

Exercise 1 – OCL (5/20 + 5 Bonus)

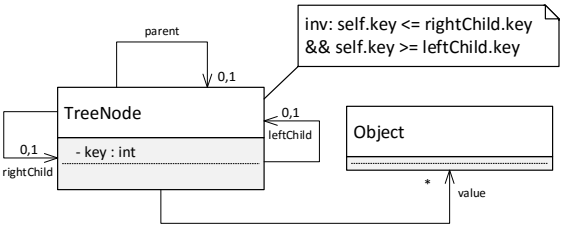

Figure 1: Class diagram for a sorted tree.

Recall the class diagram for sorted trees as shown in Figure 1.

(i) Consider the formula

$$F := \forall\, self \in allInstances_{TreeNode} \bullet key(self) \geq key(leftChild(self)) \\ \wedge\, key(self) \leq key(rightChild(self))$$

from Figure 1.

Give **system states** $\sigma_1, \sigma_2, \sigma_3$ such that

a) formula F evaluates to *true* for σ_1,
b) formula F evaluates to *false* for σ_2,
c) formula F evaluates to $\perp$ (undefined) for σ_3.

and for each system state, a (complete) **object diagram** that represents it. Also explain briefly in each case why the formula evaluates to the value you claimed. (3)

(ii) Choose one of your system states from (i) and give a **proof**, using the interpretation function as defined in the lecture, that the formula actually evaluates to the value you claimed. (2)

(iii) *Writing Proto-OCL formulae is a creative act and harder than understanding a given formula or evaluating one on a given system state. Here's a little challenge for the students in the mood to give it a try...*

Use Proto-OCL to formalise the following two requirements on the system states to be used in the software:

a) each *Object*-instance serves as 'value' for at most one *TreeNode*-instance.
b) if a *TreeNode*-instance is the *leftChild* of another *TreeNode*, then this other *TreeNode* is the *parent* of the aforementioned *TreeNode*.

Demonstrate that your submission is plausible by giving some example system states which do or do not satisfy the constraints and give the valuation of your solution on these examples. (5 Bonus)

Exercise 2 – CFA Models (10/20)

Mutual exclusion is the problem of making sure that when multiple processes or entities access a single shared resource, they do it such that only one process or entity obtains access to the shared resource at any given time.
For instance, a factory may have a single machine for a production process step that is fed by multiple incoming production lines, the machine can service only one production line at a time. We call the part of each process that performs work on the shared resource its *critical section*. Mutual exclusion thus should ensure that, at any given time, at most one process can be in its critical section.
An initial solution is the use of lock variables, that is, shared variables that indicate whether the resource is in use. A process waits until the value of the lock variable indicates that the resource is free, sets the value of the variable to indicate that the resource is locked, performs the required work and then resets the variable to indicate that the resource is free again.

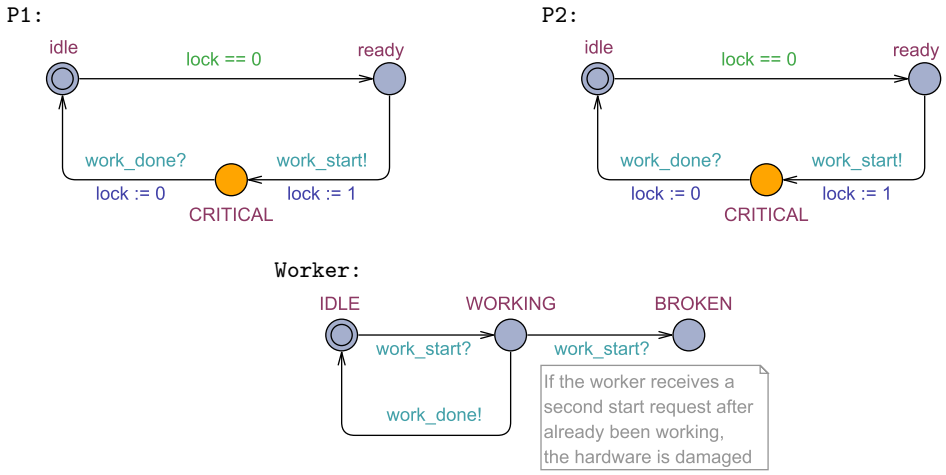

Figure 2: CFA Model $\mathcal{C}$ of mutual exclusion by using lock variables.

For the following tasks, consider the CFA model of mutual exclusion by using lock variables shown in Figure 2. The model contains two identical processes `P1` and `P2` and a shared resource `Worker`.

(i) Draw the reachable part of the **transition graph** of $\mathcal{C}$. Make sure to clearly mark the inital configuration(s). (5)

(ii) Is mutual exclusion satisfied in this model? Prove your answer by using the transition graph. (1)

(iii) Does the model have a *deadlock*? Prove your answer by using the transition graph. (1)

(iv) Use UPPAAL to simulate the model and generate a trace that demonstrates a scenario where mutual exclusion **is** satisfied. Make sure to submit your model and trace files. (2)

(v) Give a **query** to check mutual exclusion on your UPPAAL model. Use the verifier to check whether your query is satisfied. Document your results. (1)

Exercise 3 – CFA Model Verification (5/20)

A more sophisticated approach to ensure mutual exclusion is the *Bakery algorithm*[1]. It works by assigning each process requesting access to the critical section a number, much like the numbers printed on tickets for queuing at a bakery. Each process waits until no other process has a lower number. Since obtaining a number happens concurrently, two processes may obtain the same number. In that case, the bakery algorithm breaks the tie by giving priority to the process with the lowest identifier (PID).

(i) The files `bakeryA.xml`, `bakeryB.xml` and `bakeryC.xml` provide models of three different proposals to implement the Bakery algorithm. The corresponding developers are not available and have not provided convincing arguments for why their implementation is correct; we doubt that all of them are.

Analyse the three models for whether they satisfy mutual exclusion. (3)

Hint: you may use any approach you like for the analysis.

For those models which do not satisfy mutual exclusion, (construct,) save, and submit a trace file that demonstrates this fact; make sure to specify which trace file corresponds to which model. For those models which do satisfy mutual exclusion, provide a strongly convincing argumentation.

One approach to this task is to ***write a query*** *that serves to determine which one of the three models guarantees mutual exclusion and use the* UPPAAL *tool accordingly. You can determine whether process k is in its critical section by using the expression* `Process(k).CRITICAL`*.*

Explain in your own words how your query works, what do you observe with your query and how it relates to the property of mutual exclusion.

Uppaal Usage Instructions

UPPAAL is installed in the Linux machines of the computer pool. To execute it, use the following command line:

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

(ii) For the model that satisfies mutual exclusion,[2] we would like to **investigate** how the complexity of verifying that property behaves with respect to the number of processes. You can adjust the number of processes in the model by setting the value of constant `N` in the global declarations.

Measure the number of states explored and the time used for checking mutual exclusion[3] on the model configured for $N = 2, 3, 4, \ldots$ processes and plot the results (see the UPPAAL command line usage instructions below).

What is the maximum number of processes that can be analyzed on the machines of the computer pool? Analyze your graph and state a hypothesis about the complexity of verification relative to the number of processes. (2)

Make sure to keep the computing environment conditions stable between measurements. Don't forget to indicate with your results the specifications of the machine in which you performed your measurements and state by which procedure you obtained your timing measurements (to make the experiment reproducible).

[1] `https://en.wikipedia.org/wiki/Lamport%27s_bakery_algorithm`

[2] Ask your tutor for advise if you didn't manage to find one correct model in the previous task.

[3] If you followed a different approach than using UPPAAL queries in the previous task, just use the query "`A[] not deadlock`" (it checks for absence of deadlocks in the model).

Uppaal Command Line Usage Instructions

An UPPAAL command line verifier is also available in the Linux machines of the computer pool. To execute it, use the following command line:

```
/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. 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
```