Prof. Dr. A. Podelski, Dr. B. Westphal
S. Feo Arenis

Sommersemester 2016

Softwaretechnik/Software Engineering

http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl

Exercise Sheet 5

Early submission: Wednesday, 2016-07-06, 12:00 Regular submission: Thursday, 2016-07-07, 12:00

Exercise 1 – 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.

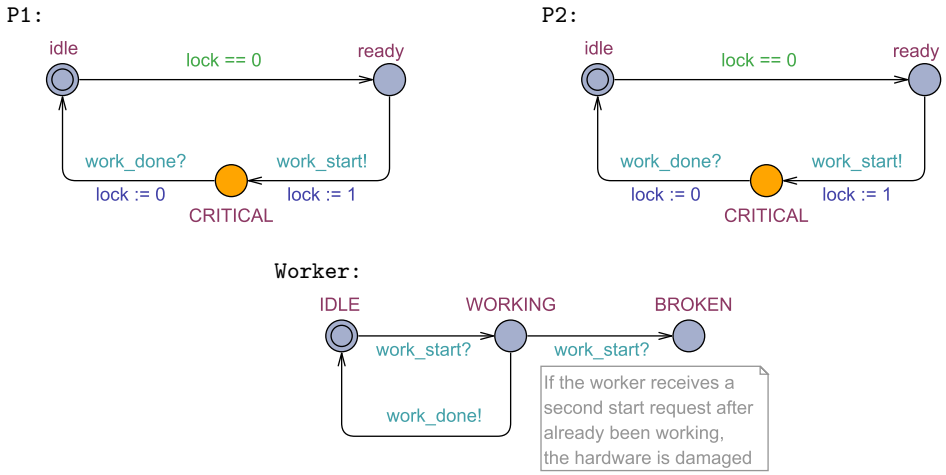

Figure 1: CFA Model $\mathcal{C}$ of mutual exclusion by using lock variables.

For the following tasks, consider the CFA model of mutual exclusion by using lock variables shown in Figure 1. 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 2 – 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) In the files `bakeryA.xml`, `bakeryB.xml` and `bakeryC.xml`, we have provided models of three possible implementations of the Bakery algorithm.

Write a query that serves to determine which one of the three models guarantees mutual exclusion. 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. (2)

(ii) Use your query to **check with Uppaal** which model does satisfy mutual exclusion. For those that do not, save and submit a trace file that demonstrates it. Make sure to specify which trace file corresponds to which model. (1)

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
```

(iii) For the model that satisfies mutual exclusion, 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 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.

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

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
```

Exercise 3 – Black-box Testing (5/20)

For this task, we have developed an implementation of the shipping costs calculator of Exercise Sheet 3. The specification for the calculation is the decision table for shipping cost calculation without the COD rule (R1) given in the tutorial slides (slide 11).

The implementation has the following specification (pre- and postcondition):

- The dimensions allowable for the package are input in whole centimeters and may range from 1cm to 250cm.
- The weight of the package is input in whole grams and may range from 1g to 32000g.
- The type of the destination address is given as a string, it can be 'metro', 'interm' or 'rural'.
- The program calculates the price in cents and prints it on the screen.

The users of the program observed some calculations being performed incorrectly. Unfortunately, they cannot remember which inputs caused the program to misbehave. You are now in the role of the test engineer and are requested to create a test suite. In particular, we are interested in determining which rules of the decision table have been implemented incorrectly.

(i) Give a **test suite** to check whether the program correctly implements all rules of the decision table. I.e., whether the program calculates the shipping price as specified when operated inside its specification. Describe the strategy you used to select the test cases.

Submit a test script according to the instructions below. (4)

(ii) Your tutor will execute the tests in your test suite using your test script. There are the following possibilities:

- Your test suite finds one rule that is implemented incorrectly. (1)
- Your test suite finds a second rule that is implemented incorrectly. (5 bonus)
- Your test suite finds a third rule that is implemented incorrectly. (100 bonus)
- Your test suite finds more rules that are implemented incorrectly. (1000 bonus each)

Hint: You can assume that the project leader allotted a time budget for the creation of the test suite equivalent to 4 exercise sheet points. Submit a number of test cases that is reasonable to achieve within that budget.

Instructions for the Testing Exercise

Before you begin, you need to request a team ID to your tutor by mail. We have created a separate binary to test for each team. Your tutor will assign a number from 0 to 99. The team ID will be used to evaluate your results on the particular implementation of the shipping calculator.
In order to create a test script, login to one of the Linux machines of the computer pool and perform the following steps:

- Create a directory where you would like to store your results.
- Execute the installation script as follows:
  `/home/arenis/testing/install.sh ID`
  where `ID` is the group ID that you received from your tutor. The script will create the file `testsuite.sh` on the current directory.
- Use your favorite editor to open that file and insert one line per test case at the end of the file using the folloowing format:

  ```
  do_test W H L Wt Type Exp
  ```

  where:

  - `W` is the width of the package in cm.
  - `H` is the height of the package in cm.
  - `L` is the length of the package in cm.
  - `Wt` is the weight of the package in grams.
  - `Type` is the type of the shipping address: 'metro', 'interm' or 'rural'.
  - `Exp` is the expected result of the test.
- Save the file and submit it.

Running the tests (optional)

Note that task (i) only requires you to submit your test script. However, if you would like to execute the test suite yourself, the binaries for testing are accessible. You may execute them at your discretion.

- To run the test cases in your test suite, execute the command:
  `./testsuite.sh`
- The script will execute your test suite and report the results.