DD2459: Software Reliability, sofRel22

Lab 3: Model-based Testing and
Automated Test-case Generation
Answer all 4 questions

Introduction:

In this lab we are going to carry out model-based testing from simple state-machine models. We will also use a powerful model checker to automatically construct test cases from test requirements. This approach represents the state-of-the-art in testing of embedded systems.

We will use a temporal logic model checker for Kripke structures known as **NuSMV** (new Symbolic Model Verifier). The tool NuSMV is a publicly available (GNU license) application that contains powerful algorithms that can be used to automatically generate test cases for embedded systems.

On the Canvas Assignment page you will also find **3 files** that you need to download besides this one. The first file "**`instructions.pdf`**" gives you the most basic information you need to get NuSMV up and running on your Ubuntu workstation. The second file contains a case study "**`bitshift.smv`**", that was presented in class, which you will need to complete and execute as one exercise. The third file contains another case study **"`carcontroller.smv`"** which was also studied in class. Using these two files and the course slides, you should have enough information to complete the lab.

Exercises.

1. Recall the **2-bit shift register** discussed in the lecture notes (Lecture 6). **UML statecharts** are a powerful design language for state machine models of systems that include *events, conditions, actions, variables* and *assignments*. Draw a model of the 2-bit shift register using these features in the UML Statechart language (you can work with UML version 2.4 or higher). **Note: in this question you cannot simply copy the diagram in my lecture notes – you must translate it into UML using an appropriate combination of UML variables, assignments, conditions, events etc.**

Hint 1: you should use a graphics tool, such as Powerpoint or any similar tool.
Hint 2: see Bibliography reference 5 below.

2. Consider the file **`bitshift.smv`**. This file is an incomplete model of the ***2-bit shift register***. In particular, the next state functions for `Bit1` and `Bit2` are *missing*. Complete these definitions. Then execute the file using the command

```
NuSMV –bmc bitshift.smv.
```

Note down carefully the output of NuSMV, and use it to confirm that your completed definitions are actually correct. Which specific output data confirms the correctness of your definitions?

3. Consider the simple ***car-controller*** discussed in the lecture notes. Download the file **`carcontroller.smv`** from the course web page. You will now generate three test cases that achieve **100% node coverage** (NC) for the CC. We want to be sure we reach each node (state). There are 3 nodes in total.

(a) Run NuSMV on the existing file **`carcontroller.smv`** with the command

```
NuSMV –bmc carcontroller.smv
```

Write down the counterexample to the 1 NC trap property which the tool generates. Clearly this achieves 33% NC!

(b) Extract from this counterexample a suitable test case, consisting of input values and output predictions. **Hint**: you should e.g. transfer the sequences of values to a table of the form

	t = 0	t = 1	t = 2	t = 3
inputs				
outputs				

Remember to store the test requirement alongside the test case for future reference.

(c) Write out 2 additional trap properties needed for 100% NC as LTL formulas so that you achieve 100% NC. Then execute the file again and repeat step (b) above to extract the 2 additional test cases. You should now have achieved 100% NC!

4. Continuing with the car-controller example, we are going to generate a set of test cases which achieve **100% edge coverage** (EC). We want to be sure we traverse each edge between any pair of nodes. There are 6 edges in total.

(a) Run NuSMV on the existing file **`carcontroller.smv`** with the command

```
NuSMV –bmc carcontroller.smv
```

Write down the counterexample to the 1 EC trap property which the tool generates. Clearly this achieves 16% EC!

(b) Extract from this counterexample a suitable test case, consisting of input values and output predictions. You should e.g. transfer the sequences of input and output values to a table as in (b) above. Remember to store the test requirement alongside the test case for future reference.

(c) Write out 5 additional trap properties needed for 100% EC as LTL formulas so that you achieve 100% EC. Then execute the file again and repeat step (b) above to extract the 5 additional test cases. You should now have achieved 100% EC!

Bibliography:

1. http://nusmv.fbk.eu/ nuSMV homepage.

2. http://en.wikipedia.org/wiki/NuSMV Wikipedia entry about nuSMV
3. http://en.wikipedia.org/wiki/Linear_Temporal_Logic Wikipedia entry about LTL
4. http://www.nada.kth.se/~karlm/sofrel/SNA-TR-2007-P2-04.pdf G. Fraser et al., Testing with Model Checkers, a Survey, 2007.
5. M. Seidel et al., *UML@classroom* – a recommended UML book which can be downloaded from KTH library https://www.amazon.co.uk/Classroom-Undergraduate-Topics-Computer-Science/dp/3319127411/ref=sr_1_1?crid=37JQOAZ42LGCM&keywords=uml+classroom&qid=1583401752&sprefix=UML%40%2Caps%2C342&sr=8-1