Testing and Verification of Emb. Sys. (DT8021) Solutions to Model Examination - May 2015

Important Notes. It is not allowed to use study material, computers, and calculators during the examination. The examination comprises 5 question in 2 pages. Please check beforehand whether your copy is properly printed. Give complete explanation and do not confine yourself to giving the final answer. The answers may be given in Swedish or English. **Good luck!**

Exercise 1 (20 points) Define the following concepts:

1. Fault, Error, Failure
2. Robust Equivalence-Class Testing
3. Regression Testing
4. All-Uses Coverage Criterion

Solution.

1. Fault is a mistake which is the result of incorrect interpretation or implementation of the requirements. Error is the wrong program state. Failure is the observable result of error in the program behavior.
2. In equivalence-class based testing, the input domain (which is the domain of a number of physical variables) is divided into a number of equivalence classes and from each class, at least three representatives are chosen for generating test-cases. To test robustness values out of the specified input domain are also provided in the test cases.
3. Regression testing refers to the process of re-testing the evolved versions of the software while maintaining the old test cases and adding new test cases.
4. All-Uses criterion specifies that a test-set is adequate if the set of paths covered by the test-set contains at least one DC past for each use of each defined value for each variable.

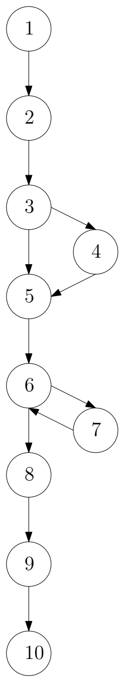
Exercise 2 (25 points) Consider the following program.

```
 1: read(x);
 2: read(y);
 3: if x < 10 then
 4:     x := 10;
 5: end if
 6: while y < x then
 7:     y := y + 1;
 8: end while
 9: x := x + 1
10: write(x);
```

1. Draw the control-flow graph of the program (5 pts),
2. Calculate all prime paths of the CFG (10 pts),
3. Define a set with the fewest number of test cases that satisfies the all-prime-path coverage criterion (10 pts).

Solution.

1. The CFG of the program is given below:



2. Prime paths (simple paths that are not included in any other simple path) are given below:

   [1, 2, 3, 5, 6, 8, 9, 10]

   [1, 2, 3, 4, 5, 6, 8, 9, 10]

   [1, 2, 3, 5, 6, 7]

   [1, 2, 3, 4, 5, 6, 7]

   [7, 6, 8, 9, 10]

   [6, 7, 6]

   [7, 6, 7]

3. The following set of two test cases satisfies the all-prime-path criterion:

   inputs: x = 9, y = 12, expected output x = 11, covers: [1, 2, 3, 4, 5, 6, 8, 9, 10]

   inputs: x = 11, y = 12, expected output x = 12, covers: [1, 2, 3, 5, 6, 8, 9, 10]

   inputs: x = 9, y = 6, expected output x = 11, covers: [1, 2, 3, 5, 6, 7], [7, 6, 8, 9, 10], [6, 7, 6] and [7, 6, 7]

   inputs: x = 12, y = 6, expected output x = 13, [1, 2, 3, 4, 5, 6, 7]

Exercise 3 (20 points) Explain the meaning of the following formulas in English.

1. `E <> deadlock` (5 pts),
2. `A <> (a.l or a.lp) and v <= 2` (5 pts),
3. `a.l --> a.lp` (10 pts).

Solution.

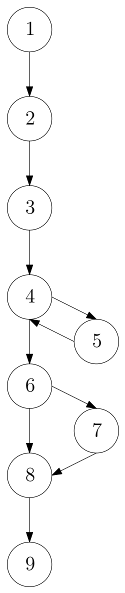
1. There is some execution in which some state does not have any outgoing enabled transition,
2. For all executions eventually a state is reached in which process (automaton instance `a` is either in location `l` or in location `lp` and the value of global variable `v` is at most 2,
3. For all execution if process `a` is in location `l`, then eventually in all future executions `a` will be in location `lp`.

Exercise 4 (25 points) Calculate *Slice*$(9, \{x\})$ for the following program. The final solution is not sufficient; elaborate on the steps towards the final solution. (20 pts)

```
1: read(x);
2: read(y);
3: z := y;
4: while y < 10 then
5:     z := z + 1;
6: if z < y then
7:     x := 10;
8: x := x + 1
9: write(x);
```

Is the calculated slice optimal? Motivate your answer. (5 pts)

Solution. The flow graph of the program is depicted below.



Based on this CFG, the slice $Slice(9, \{x\})$ is calculated using the following three approximations.

m	**DEF(m)**	$\mathbf{Relevant_0(m)}$	$\mathbf{Slice_0}$	$\mathbf{Cond_1}$	$\mathbf{Rel_1}$	$\mathbf{Slice_1}$	$\mathbf{Cond_2}$	$\mathbf{Rel_2}$	$\mathbf{Slice_2}$
1	$\{x\}$	$\emptyset$	$\surd$	$\times$	$\emptyset$	$\surd$	$\times$	$\emptyset$	$\surd$
2	$\{y\}$	$\{x\}$	$\times$	$\times$	$\{x\}$	$\surd$	$\times$	$\{x\}$	$\surd$
3	$\{z\}$	$\{x\}$	$\times$	$\times$	$\{x, y\}$	$\surd$	$\times$	$\{x, y\}$	$\surd$
4	$\emptyset$	$\{x\}$	$\times$	$\times$	$\{x, y, z\}$	$\times$	$\surd$	$\{x, y, z\}$	$\surd$
5	$\{z\}$	$\{x\}$	$\times$	$\times$	$\{x, y, z\}$	$\surd$	$\times$	$\{x, y, z\}$	$\surd$
6	$\emptyset$	$\{x\}$	$\times$	$\surd$	$\{x, y, z\}$	$\surd$	$\surd$	$\{x, y, z\}$	$\surd$
7	$\{x\}$	$\emptyset$	$\surd$	$\times$	$\emptyset$	$\surd$	$\times$	$\emptyset$	$\surd$
8	$\{x\}$	$\{x\}$	$\surd$	$\times$	$\{x\}$	$\surd$	$\times$	$\{x\}$	$\surd$
9	$\emptyset$	$\{x\}$	$\times$	$\times$	$\{x\}$	$\times$	$\times$	$\{x\}$	$\times$

No, the calculated slice is not optimal since in the case of y $<$ 10, the loop does not terminate and hence, the rest of the slice does not contribute to the value of x. However, termination of programs for most programming languages is undecidable and hence calculating an optimal slice cannot be decided algorithmicly.

Exercise 5 (10 points) Consider the following procedure, which is supposed to take an array of integers and its size and write the average of the numbers in the array (up to size) on the screen. It turns out that it outputs ‘8.0’ when input *arr*= [2, 2, 2, 14, 8] is given (which is incorrect). Simplify the test-case using simplification. (Assume that you can manually check whether the outcome of each test is really correct or not.)

```
const MAX = 100;
procedure writeAvg(arr: array of integer);
var i : integer;
   avg : real;
begin
   avg := 0;
   for i := 0 to size(arr) - 1 do
      avg := avg + arr[size(arr) - 1];
   write(avg/size(arr));
end;
```

Solution. First we decompose the test-case into two almost equal pieces, e.g., [2, 2, 2] and [14, 8]. It turns out that [2, 2, 2] passes and [14, 8] fails. Hence, we iterate the test case minimization algorithm with [14, 8] as the new (smaller) failing test case. Decomposing [14, 8] into two equal pieces results in two passing test-cases and since [14, 8] cannot be decomposed any further, the algorithm terminates with [14, 8] as the minimal failing test-case.

$$\begin{array}{l} ddmin([2,2,2,14,8],2) = \\ ddmin([14,8],2) = \\ 14,8 \end{array}$$