Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Matteo Rossi and Damian Tamburri

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering 2 – Written Exam 1 (WE1)

June 10th, 2022

Notes

1. Remember to write your name and Id number (matricola) on each piece of paper that you hand in.
2. You may use a pencil.
3. Incomprehensible handwriting is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden, with the exception of an ebook reader.
5. The exam is composed of three exercises. Read carefully all points in the text!
6. **Total available time for WE1: 1h and 30 mins**

Question 1 Alloy (8 points)

We want to describe a system that distributes the computation of functions across nodes of a network. Nodes can be of different capacities: normal, small, and tiny. Nodes execute functions, where each function has a name and it can require a heavy, normal, or light computation. A node can execute multiple instances of the same or of different functions; also, different instances of the same function can be executed by different nodes. A normal node can execute functions that require a heavy or normal computation; a small node can execute functions that require a normal or a light computation; a tiny node can only execute functions that require a light computation.

Alloy_1 (4 points)
Define a set of signatures, and associated constraints, that capture the elements described above.

Alloy_2 (1 point)
Define a predicate `isCompatible` that takes as input the instance of a function and a node, and returns true if the computation executed by the function instance is compatible with the capacity of the node.

Alloy_3 (3 points)
Define a predicate that specifies the behavior of an operation `moveFunction` that takes as input two nodes *n1* and *n2* and the name of a function *f*, and moves all instances of executions of function f from *n1* to *n2*.

Solution

Alloy_1

```
sig FunctNames {}

abstract sig NodeType {}
one sig NormalNode extends NodeType {}
one sig SmallNode extends NodeType {}
one sig TinyNode extends NodeType {}

abstract sig CompType {}
one sig HeavyComp extends CompType {}
one sig NormalComp extends CompType {}
one sig LightComp extends CompType {}

sig Function {
  name : FunctNames,
  comp : CompType
}

sig FunctInstance {
  funct : Function
}


fact functionConstraint {
  all disj f1, f2: Function | not f1.name = f2.name
}

sig Node {
  type : NodeType,
```

```
  exec : set FunctInstance
}{
  type = NormalNode implies exec.funct.comp & LightComp = none
  type = SmallNode implies exec.funct.comp & HeavyComp = none
  type = TinyNode implies exec.funct.comp in LightComp
}
```

Alloy_2

```
pred isCompatible (fi: FunctInstance, n: Node){
  fi.funct.comp = HeavyComp implies n.type = NormalNode
  fi.funct.comp = NormalComp implies not n.type = TinyNode
  fi.funct.comp = LightComp implies not n.type = NormalNode
}
```

Alloy_3

```
pred moveFunction (n1, n2, n1', n2': Node, f: FunctNames){
  // pre-conditions
  f in n1.exec.funct.name
  (all fi : FunctInstance | (fi.funct.name = f and fi in n1.exec) implies
                            isCompatible[fi, n2])
  // post-conditions
  let funcs = { fi : FunctInstance | fi.name = f and fi in n1.exec } |
    ( n1'.type = n1.type         and
      n1'.exec = n1.exec - funcs and
      n2'.type = n2.type         and
      n2'.exec = n2.exec + funcs )
}
```

Question 2 Testing (5 points)

Consider the following fragment of code:

```
0  int myfunct(int x, int y){
1    int n, count;
2    if (x < 0 || y < 0 || x == y)
3        return 1;
4    if (x > y) {
5        y = x*2;
6    } else {
7        n = 1;
8    }
9    count = 0;
10   while (x < y){
11       n = n+1;
12       x = x+2;
13   }
```

```
14   return n;
```

T_1 (3 points)
Identify the def-use pairs for program `myfunct` and say whether they highlight anything unusual.

T_2 (2 points)
If def-use pairs highlight potential problems, use symbolic execution to argue whether the problems can actually occur or not.

Solution

T_1

x: <0,2>, <0,4>, <0,5>, <0,10>, <0,12>
<12,10>, <12,12>
y: <0,2>, <0,4>, <0,10>
<5,10>
n: <7, 11>, <7, 14>, <11, 11>, <11, 14>
<?,11>, <?,14>
count: <9, ?>

Variable *n* is potentially used at lines 11 and 14 without having been previously initialized.
Variable *count* is set at line 9, but it is then never used, which is not necessarily a problem, though it is not ideal.

T_2

As mentioned above, the fact that *count* is initialized, but it is never used, is an imperfection, but it is not necessarily a problem. On the other hand, we need to investigate whether it is possible to reach line 11 without executing line 7 first. Then, we need to determine whether the following path is feasible: 0, 1, 2, 4, 5, 9, 10, 11.

Using symbolic execution, we obtain the following path condition:

Path 0, 1, 2, 4, 5, 9, 10, 11

0: x = X, y = Y	PC: true
1:	PC: true
2:	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$
4:	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$ and $X > Y$
5: y = X*2	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$ and $X > Y$
9: count = 0	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$ and $X > Y$
10:	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$ and $X > Y$ and $X < X*2$
11:	PC: $X \geq 0$ and $Y \geq 0$ and $X \neq Y$ and $X > Y$ and $X < X*2$

In conclusion, the path is possible, it is enough to have that, initially, x is greater than y. For example, the inputs x = 4, y = 3 would execute the problematic path.

Question 3 Availability (3 points)

Consider a workflow made of the following 4 steps:

1. image acquisition
2. image recognition of feature 1
3. image recognition of feature 2
4. decision

where steps 2 and 3 (the recognition of specific features of the image) are executed in parallel, but the results of both steps are necessary to take the final decision.
We have several components that can perform the various steps of the workflow. In particular, step 1 is executed by components whose availability is 95%; step 2 is executed by components whose availability is 80%; step 3 is executed by components whose availability is 90%; step 4 is executed by components whose availability is 99%.

Design an architecture of the system whose availability is 99%.

Solution

Although steps 2 and 3 are executed in parallel, since both of their results are necessary for the execution of step 4, the workflow is to be considered, from the availability point of view, a series of the 4 steps.
Since we need an overall availability of 99%, and each component has an availability of 99% or less, each step must be performed by a set of replicated components, so that it has an availability that is greater than 99% (the series decreases the overall availability).
In particular, we have the following:

1. The parallel of 2 components for step 1 achieves an availability, for that step, of 99.7%.
2. The parallel of 4 components for step 2 achieves an availability, for that step, of 99.8%.
3. The parallel of 3 components for step 3 achieves an availability, for that step, of 99.9%.
4. The parallel of 2 components for step 4 achieves an availability, for that step, of 99.99%.

All in all, the resulting total availability is 0.997*0.998*0.999*0.9999 = 0.994, which is enough.
Note that if we decide to have only one component for step 4, we have: 0.997*0.998*0.999*0.99 = 0.985, which is not enough, instead.