SAMPLE EXAM
Testing, Debugging, and Verification
TDA567/DIT082

—

Extra aid:	Only dictionaries may be used. Other aids are *not* allowed!
Grade intervals:	**U**: 0 – 23p, **3**: 24 – 35p, **4**: 36 – 47p, **5**: 48 – 60p, **G**: 24 – 47p, **VG**: 48 – 60p, **Max.** 60p.

Please observe the following:

- This exam has 18 numbered pages.
  Please check immediately that your copy is complete
- Answers must be given in English
- Use page numbering on your pages
- Start every assignment on a fresh page
- Write clearly; unreadable = wrong!
- Fewer points are given for unnecessarily complicated solutions
- Indicate clearly when you make assumptions that are not given in the assignment

Good luck!

Assignment 1 (Testing) (12p)

(a) **Briefly** describe the main features of the *Extreme Testing* methodology. Also list **three** of its main advantages.

(b) Construct a minimal set of test-cases for the code snippet below, which satisfy *condition coverage*

```
if (a > b || b < 0)
   return a;
else
   return b;
```

(c) We discussed another two criteria for logic coverage in class. Describe these. Also describe the relationship between the three logic based criteria. Does your test-set from (b) satisfy any of the other coverage criteria. Why/why not?

Solution
[5p, 2p, 5p]

(a) Extreme testing is a development methodology where test-cases must be developed *before* the code is written. These tests must be re-run after every incremental code change.

Advantages: *(3 marks for three sensible advantages, e.g. something like the below. No extra marks for more than 3. If more than 3 included and something is totally wrong, there might be penalties.)*

- Clear idea of what program should do before starting coding. Test-cases provide a specification for the program.
- Facilitates regression testing / integrates regression testing in development process.
- May start with simple design and incrementally optimise later, without risking breaking the specification. Therefore may have rapid development process.

(b)
Need two test-cases, for instance
`{a --> 2, b --> -1}` and `{a --> 2, b --> 3}`

(c)
This question checks if they have understood the logic based criteria. Full marks only if they give a satisfactory explanation to whether their answer to (b) satisfy any of the other criteria (mine above does not, as in both cases the decision comes out true). In addition, they should also give the following definitions of decision coverage and MCDC:

Decision Coverage (DC) For a given *decision* d, DC is satisfied by a test suite TS if it contains at least two tests, one where d evaluates to *false*, and one where d evaluates to *true*. For a given *program* p, DC is satisfied by TS if it satisfies DC for all $d \in D(p)$.

Modified Condition Decision Coverage (MCDC) For a given *condition* c in decision d, MCDC is satisfied by a test suite TS if it contains at least two tests, one where c evaluates to *false*, one where c evaluates to *true*, d evaluates differently in both, and the other conditions in d evaluate identically in both. For a given *program* p, MCDC is satisfied by TS if it satisfies MCDC for all $c \in C(p)$.

- DC and CC are orthogonal (i.e. neither subsume the other)
- MCDC subsumes DC and CC.

Assignment 2 (Debugging) (11p)

(a) When is a statement B control dependent on a statement A?

(b) In the below small Dafny program, on which statement(s) is/are the statements in line 9 data dependent?

```
1 method M(n : nat) returns (b : nat){
2       if(n == 0)
3          { return 0; }
4       var i := 1;
5       var a := 0;
6       b := 1;
7       while (i < n)
8       {
9        a, b := b, a+b;
11       i := i +1;
12      }
13 }
```

(c) On which statements is line 11 *backward dependent*?

(d) The `ddMin` algorithm computes a minimal failure inducing input sequence. It relies on having a method `test(i)` which returns `PASS` if the input `i` passes the test or `FAIL` if the `i` causes failure (i.e. bug is exhibited).
Suppose our input consists of representations of DNA sequences, made out out the letters `G,A,T,C` which represent the nucleotides. Let `test` return `FAIL` whenever the DNA sequence contains *two consecutive occurrences of the letter C* somewhere in the sequence.
Simulate a run of the `ddMin` algorithm and compute a minimal failing input from an initial failing input `[G,T,A,C,C,A,G,C]`. Clearly state what happens at *each step* of the algorithm and what the final result is. Correct solutions without explanation will not be given the full score.

Solution
[1*p*, 2*p*, 2p, 6p]

(a)
B is control dependent on A, if B's execution is potentially controlled by A.

(b)
Line 9 is data-dependent on lines 5 and 6 as well as itself, as it may reads the values of the previous iteration. 1 mark for lines 5 and 6, 2 marks if they also have line 9 in the answer.

(c)
Line 11 is backward dependent on lines 4 and 7 for the first iteration of the loop. On repeated iterations of the loop, it is backward dependent on lines 7 and on itself.

(d)
Start with granularity $n = 2$ and sequence `[G,T,A,C,C,A,G,C]`.

The number of chunks is 2
==> $\texttt{n} : 2, [\texttt{C}, \texttt{A}, \texttt{G}, \texttt{C}]$ PASS (take away first chunk)
==> $\texttt{n} : 2, [\texttt{G}, \texttt{T}, \texttt{A}, \texttt{C}]$ PASS (take away second chunk)

Increase number of chunks to $min(n * 2,\ len([\texttt{G}, \texttt{T}, \texttt{A}, \texttt{C}, \texttt{C}, \texttt{A}, \texttt{G}, \texttt{C}])) = 4$
==> $\texttt{n} : 4, [\texttt{A}, \texttt{C}, \texttt{C}, \texttt{A}, \texttt{G}, \texttt{C}]$ FAIL (take away first chunk)

Adjust number of chunks to $max(n - 1, 2) = 3$
==> $\texttt{n} : 3, [\texttt{C}, \texttt{A}, \texttt{G}, \texttt{C}]$ PASS (take away first chunk)
==> $\texttt{n} : 3, [\texttt{A}, \texttt{C}, \texttt{G}, \texttt{C}]$ PASS (take away second chunk)
==> $\texttt{n} : 3, [\texttt{A}, \texttt{C}, \texttt{C}, \texttt{A}]$ FAIL (take away third chunk)

Adjust number of chunks to $max(n - 1, 2) = 2$
==> $\texttt{n} : 2, [\texttt{C}, \texttt{A}]$ PASS (take away first chunk)
==> $\texttt{n} : 2, [\texttt{A}, \texttt{C}]$ PASS (take away first chunk)

Increase number of chunks to $min(n * 2,\ len([\texttt{A}, \texttt{C}, \texttt{C}, \texttt{A}]) = 4$
==> $\texttt{n} : 4, [\texttt{C}, \texttt{C}, \texttt{A}]$ FAIL (take away first chunk)

Adjust number of chunks to $max(n - 1, 2) = 3$
==> $\texttt{n} : 3, [\texttt{C}, \texttt{A}]$ PASS (take away first chunk)
==> $\texttt{n} : 3, [\texttt{C}, \texttt{A}]$ PASS (take away second chunk)
==> $\texttt{n} : 3, [\texttt{C}, \texttt{C}]$ FAIL (take away third chunk)

Adjust number of chunks to $max(n - 1, 2) = 2$
==> $\texttt{n} : 2, [\texttt{C}]$ PASS (take away first chunk)
==> $\texttt{n} : 2, [\texttt{C}]$ PASS (take away second chunk)

As $n == len([\texttt{C}, \texttt{C}])$ the algorithm terminates with minimal failing input $[\texttt{C}, \texttt{C}]$

Assignment 3 (Formal Specification) (15p)

For this question, we will consider a part of a program for an autonomous robot moving around exploring a cave where there are both treasures and hungry monsters which will eat any human (but it won't eat robots of course). The robot's job is to completely map the cave before the humans move in to collect the treasures, so they can avoid any rooms with monsters.

The environment in which the robot moves around is represented by a matrix of integer values between 0-3 where these values indicate the following possible statuses of each location:

0: The location is not yet visited.

1: The location has been visited.

2: The location has been visited and sensory data indicate that there is a treasure here!

3: The location has been visited and sensory data indicate that there is a a hungry monster here!

The robot moves around one step at the time, until it has explored the whole world. When the whole cave system is explored, the robot teleports back to its starting position. Teleportation requires a lot of power, so the robot can only do it once, then it has to recharge, so it is important that this only happens once it has finished its exploration task.

The robot has an internal model of the cave system, represented by the Dafny class `Q3` below. To declare a matrix in Dafny, we use the type `array2`, which represents a two dimensional array. To access the width and height of a two dimensional array `a` in Dafny we use `a.Length0` and `a.Length1` respectively. To access an element in a two dimensional array, we write `a[i,j]`.

Continued on next page!

```
class Q3 {

  var world : array2<int>;      // The robots internal map of the cave system
  var x : int, y : int;        // Current x,y co-ordinates

   // Holds if this object is valid: i.e. x,y co-ordinates are
  // in scope and the world map contains only allowed values.
  predicate Valid()
  reads this;
  {  }

   // Holds if location (x1,y1) is a valid location in this world.
  predicate AllowedLoc(x1 : int, y1 :int)
  reads this;
  { }

  // Holds if the whole world is explored.
  predicate CompletelyExplored()
  reads this;
  {  }

  method Init(w : int, h : int)
    {
    world := new int[w,h];
    forall(i,j | 0 <= i < w && 0 <= j < h)
      { world[i,j] := 0; }    // nothing visited.
    x,y := 0,0;               // start location
    world[0,0] := 1;          // start location is visited.
  }

  // Robot reading its sensors and updating current location accordingly.
  //Input==1 means current location is empty
  //Input==2 means there is a treasure here!
  //Input==3 menas there is a monster here!
  method ProcessSensorReading(input: int)
  { }

 // Robot moves one step south (i.e it decreases its y-coordiante by 1),
 // provided that is possible.
 // If the new location has not been visited before, the robot should update it
 // and set its status as "visited" (i.e to 1).
 // If the new location has been visited, its status should remain the same.
   method MoveSouth()
  { }

  // Robot teleports back to the starting position (0,0),
  // provided that it has finished exploring the cave.
  method TeleportHome()
  { }
}
```

Continued on next page!

The robot is very expensive, it is important that the source code is correct. Your task is to enrich the class `Q3` with specifications for the methods predicates.

(a) Complete the specifications and implementations of the three predicates: `Valid()`, `AllowedLoc(x1,y1)` and `CompletelyExplored`.

(b) Complete the `Init()` method with a contract specifying its pre and post conditions. In particular, `world` must be a newly allocated object, and no locations except (0,0) are not visited. Of course, `w` and `h` must have sensible values, and the object should be initialised accordingly.

(c) Write down specifications and implementations for the methods `MoveSouth` (you do not have to consider moving north, east or west, as these methods are analogous), `ProcessSensorReading` and `TeleportHome`. Take into account the informal specifications written in the comments above.

Solution
[5p, 4p, 6p]

```
class Q3 {

  var world : array2<int>;
  var x : int;
  var y : int;

  predicate Valid()
  reads this;
  {
    world != null &&
    0 <= x < world.Length0 &&
    0 <= y < world.Length1 &&
    forall i,j :: 0 <= i < world.Length0 && 0 <= j < world.Length1 ==>
    0 <= world[i,j] <= 3
  }
  predicate AllowedLoc(x1 : int, y1 :int)
  requires Valid();
  reads this;
  {
    0 <= x1 < world.Length0 &&
    0 <= y1 < world.Length1
  }
  predicate CompletelyExplored()
  requires Valid();
  reads this;
  {
    forall i,j :: 0 <= i < world.Length0 && 0 <= j < world.Length1 ==>
    0 < world[i,j] <= 3
```

```
 }

 method Init(w : int, h : int)
 modifies this;
 requires w > 0 && h > 0;
 ensures Valid();
 ensures fresh(world);
 ensures world.Length0 == w && world.Length1 == h;
 ensures x == 0 && y == 0;

 // there are many ways of expressing the below...
 ensures forall i,j :: 0 <= i < w && 0 <= j < h  && !(i==j==0) ==>
world[i,j]==0;
 ensures world[0,0] == 1;
 //Alt. to the above, might also be others
 ensures forall i,j :: 0 <= i < w && 0 <= j < h ==> (i==x && j==y &&
world[i,j] == 1) || world[i,j]==0;
 {
   world := new int[w,h];
   forall(i,j | 0 <= i < w && 0 <= j < h)
   {
     world[i,j] := 0;  //nothing visited.
   }
   x,y := 0,0; // start location
   world[0,0] := 1;
 }

 method moveSouth()
 modifies `x, `y, world; // also OK: modifies this.
 requires Valid() && AllowedLoc(x,y-1);
 ensures x== old(x) && y == old(y)-1 && Valid();
 ensures old(world[x,y-1]) > 0 ==> world[x,y] == old(world[x,y-1]);
 ensures old(world[x,y-1]) == 0 ==> world[x,y] == 1;
 {
   y := y -1;
   if (world[x,y] == 0)
       { world[x,y] := 1;}
 }

 // Robot reading its sensors.
 //Input==1 means current location is empty
 //Input==2 means there is a treasure here!
 //Input==3 menas there is a monster here!
 method ReadSensor(input: int)
 modifies world;
```

```
  requires Valid();
  requires input ==1 || input == 2 || input == 3;
  ensures world[x,y] == input;
  {
    world[x,y] := input;
  }

  method TeleportHome()
  modifies ‘x, ‘y;
  requires Valid();
  requires CompletelyExplored();
  ensures x==0 && y == 0;
  {
    x,y := 0,0;
  }

}
```

Assignment 4 (Specification and Test Generation) (6p)

```
method Min(arr : array<int>) returns (min : int)
requires arr !=null && arr.Length > 0;
ensures ?
{
  var i := 1;
  min := arr[0];
  while(i < arr.Length)
  {
    if(arr[i] < min)
      {min := arr[i];}
    i := i +1;
  }

}
```

(a) Complete the above Dafny program which is supposed to compute the minimum of an array. Your answer should state suitable postconditions and loop invariants.

(b) When generating tests from specifications, it is normally required that the generated test inputs satisfy certain parts of the program specification. Which parts?

Solution
[5p, 1p]

(a)

```
method Min(arr : array<int>) returns (min : int)
requires arr !=null && arr.Length > 0;
ensures forall i :: 0 <= i < arr.Length ==> min <= arr[i];
ensures exists i :: 0 <= i < arr.Length && min == arr[i];
{
  var i := 1;
  min := arr[0];
  while(i < arr.Length)
  invariant 0 < i <= arr.Length;
  invariant forall j :: 0 <= j < i ==> min <= arr[j];
  invariant exists j :: 0 <= j < i && min == arr[j];
  {
    if(arr[i] < min)
    {min := arr[i];}
    i := i +1;
  }
}
```

(b)
The preconditions, here `arr !=null && arr.Length > 0`

Assignment 5 (Verification) (16p)

The next question is about the following little Dafny program which implements a mathematical "trick":

- Think of a number n.
- Double the number.
- If the result is negative, add 6 to the result. Otherwise, add 4 to the result.
- Half the result.
- Subtract the number you were thinking of from the result.
- Your result it either 2 or 3. If you were thinking of a positive number, it is 2, a negative and it is 3.

In Dafny:

```
method Trick(n : int) returns (m : int)
ensures n < 0 ==> m == 3;
ensures n >= 0 ==> m == 2;
{
  m := n * 2;
  if (m >= 0)
    { m := m + 4;}
  else
    { m := m + 6;}
  m := m / 2;
  m := m - n;
}
```

(a) Prove that the method `Trick` is correct using the weakest precondition calculus.

Continued on next page!

The rest of this question concerns a program with a loop, which computes x^y:

```
function exp(n : nat, m : nat) : nat
{
  if (m==0) then 1 else n * exp(n,m-1)
}

method Exp(x : nat, y : nat) returns (res : nat)
ensures res == exp(x,y);
{
  var i := 0;
  res := 1;
  while(i < y)
  {
    res := res * x;
    i := i+1;
  }
}
```

(b) Give a loop *invariant* and a loop *variant* for the loop in the Exp method above. You may want to use the recursive function provided. Note that the inputs are of type nat, so they cannot be negative.

(c) Prove the Exp method correct using the weakest precondition calculus.

Solution

[7p, 2p, 7p]

(a)

```
R: n < 0 ==> m == 3 && n >= 0 ==> m == 2;

Apply a number of Seq-rules:
wp(m := n * 2,
        wp(if (m >= 0) . . else . .,
                wp(m := m/2,
                        wp(m:= m-n, R)))))

Apply Assignment
wp(m := n * 2,
        wp(if (m >= 0) . . else . .,
                wp(m := m/2, n < 0 ==> m-n==3 && n>=0 ==> m-n==2)

Apply Assignment
wp(m := n * 2,
        wp(if (m >= 0) . . else . .,
                n < 0 ==> m/2-n==3 && n>=0 ==> m/2-n==2)
```

```
Apply Conditional rule:
wp(m := n * 2,
        m >= 0 ==> wp (m:=m+4, n < 0 ==> m/2-n==3 && n>=0 ==> m/2-n==2)
        &&
        m < 0 ==> wp (m:=m+6, n < 0 ==> m/2-n==3 && n>=0 ==> m/2-n==2)

Apply Assignment to each side:
wp(m := n * 2,
        m >= 0 ==> (n < 0 ==> (m+4)/2-n==3 && n>=0 ==> (m+4)/2-n==2)
        &&
        m < 0 ==> (n < 0 ==> (m+6)/2-n==3 && n>=0 ==> (m+6)/2-n==2)

Apply Assignemnt:
n*2 >= 0 ==> (n < 0 ==> (n*2 + 4)/2-n==3 && n>=0 ==> (n*2 + 4)/2 -n==2)
&&
n*2 < 0 ==> (n < 0 ==> (n*2 + 6)/2-n==3 && n>=0 ==> (n*2 + 6)/2-n==2)

Simplify (first conjunct implies n is positive, second, n is negative)
n*2 >= 0 ==> (false ==> . . ) && n>=0 ==> (n*2 + 4)/2 == 2+n)
&&
n*2 < 0 ==> (n < 0 ==> (n*2 + 6)/2 == 3+n && (false ==> . .)

Simplify
n*2 >= 0 ==> true && n>=0 ==> (n+2 == 2+n)
&&
n*2 < 0 ==> n < 0 ==> (n + 3) == 3+n && true

True && True
```

b)
Invariant: `res == exp(x,i) && i <= y`
Variant: `y-i`

c)
We follow the five steps from the lecture notes:

```
I : res == exp(x,i) && i <= y
V: y-i
R: res == exp(x,y)
S1: i := 0; res := 1;
S: res := res * x; i := i+1;


1) Show that the invariant holds before entry of loop:
wp(S1, I)
wp(i:=0; res :=, res == exp(x,i) && i <= y)
        Apply seq, then assignment twice
1 == exp(x, 0) && 0 <= y
        simplify
1 == 1 && true


2) Show that the loop invariant hold on each iteration.
I && B ==> wp(S, I)

I && (i < y) ==> wp(res := res * x; i := i+1, I)
        apply seq, then assignment twice

res == exp(x,i) && i <= y  && (i < y) ==> res*x == exp(x, i +1) && i+1
 <= y
        simplify using: exp(x, i+1) = x * exp(x, i) and i + 1 <= y
 implies that i < y

res == exp(x,i) && i <= y  && (i < y) ==> res*x == x * exp(x,i) && i <
 y
        simplify: divide equation by x

res == exp(x,i) && i <= y  && (i < y) ==> res == exp(x,i) && i < y

        which follows from the invariant

3) Show that the post-condition is implied when the loop exits:
I && !B ==> R

res == exp(x,i) && i <= y && !(i < y) ==> res == exp(x,y)
        Neg. loop guard !(i < y) and invariant (i <= y) imply that i==y
res == exp(x,y) ==> res == exp(x,y)
        which is trivially true


4) Show that the variant is bounded from below by 0:
I && B ==> V > 0

res == exp(x,i) && i <= y && i < y ==> y-i > 0
        As i is less than y, and y is positive (it is a natural number)
  then y-1 > 0 is true.

5) Show that the variant decrease at each loop iteration:
I && B ==> wp(V1 := V; S, V < V1)
        Apply seq rule.
```

```
I && B ==> wp(V1 := y-i, wp(S, y-i < V1)
        Apply assignments in S, setting i := i+1
I && B ==> wp(V1 := y-i, y-(i+1) < V1)
        Apply assignment
I && B ==> y-i-1 < y-i)
        Trivially true.
```

(total 60p)