EXAM
Testing, Debugging, and Verification
TDA567/DIT082

DAY: 09 January 2018 TIME: $08^{30} - 12^{30}$

Responsible:	Srinivas Pinisetty (Lecturer), Wolfgang Ahrendt (examiner)
Contact:	Srinivas Pinisetty (0733873221) Exam room visit around 9:30 and 11:30
Results:	Will be published mid February (or earlier)
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 12 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.
- Read all parts of the assignment before starting to answer the first question.
- Indicate clearly when you make assumptions that are not given in the assignment.
- **Weakest pre-condition rules are provided in Page 12**.

Good luck!

Assignment 1 (Testing) (16p)

(a) **Briefly** explain what *White Box* and *Black Box* testing is, and how they differ. (2p)

(b) You work for a company that makes tiny robots. Your employer is considering using a new third party library for taking sensor readings. However, your boss is not convinced that this library has been tested to a sufficiently high standard and asks you to run some additional tests before deciding to adopt this new third party library. You don't have access to the library source code, only specifications of the methods. Describe the *methodology* you would use to *systematically* derive test-cases from the specification. (2p)

(c) **Briefly** describe the main features of the *Extreme Testing* methodology. Also list **two** of its main advantages. (2p)

(d) Write down **two** test-cases for the small program below. Your test cases should satisfy *decision coverage* for the program. (2p)

```
int method1(int x, int y)
{
        int res = 0;
        if((x == 0) || (x > y))
                res = y;
        if (isEven(x))
                res = x/2;
        return res;
}
```

(e) For method1, does your test-suite from question (d) satisfy *condition coverage*? Motivate and explain why or why not. (2p)

(f) We also discussed about *statement coverage*, and *branch coverage* criteria that are control-flow graph based. For method1, does your test suit from question (d) satisfy *statement coverage* and *branch coverage*. Motivate and explain why or why not. (2p)

(g) Construct a minimal set of test-cases for the code snippet below, which satisfy *Modified Condition Decision Coverage*. (4p)

```
int method2(int a, int b, int c)
{
        if ( (a < 3) || (b > c && c == 5) )
                return a;
        else
                return c;
}
```

Assignment 2 Debugging: Minimization using DDMin (7p)

Consider a method that takes an array of integers as input, and computes a code that it returns as a result. The method *fails* if the input array consists of *two identical even numbers*. For example, the method fails when the input array is $[1, 2, 8, 6, 6, 2, 8, 5]$, $[2, 6, 7, 7, 5, 2]$.

(a) A *1-minimal failing input* is an input where if you remove any single element, the resulting input succeeds. List *all* 1-minimal failing inputs in the following input array: `[1,2,8,6,6,2,8,5]`. (2p)

(b) Simulate a run of the `ddMin` algorithm and compute a 1-minimal failing input from the following initial failing input: `[1,2,8,6,6,2,8,5]`. Clearly state what happens at *each step* of the algorithm and what the final result is. (5p)

Assignment 3 (Debugging: Backward dependencies) (7p)

The following Java method is intended to compute the minimum and maximum values occurring in an integer array.

```
 1  static void minMax(int [] a) {
 2    int min = 0;
 3    int max = 0;
 4    for(int i = 0; i < a.length; i++){
 5      if(a[i] < min)
 6        min = a[i];
 7      if(a[i] > max)
 8        max = a[i];
 9    }
10    System.out.println(‘‘Min: ’’ + min + ‘‘\n Max: ’’ + max);
11  }
```

For an input array `[1,2,3]`, the program outputs:

```
Min: 0
Max: 3
```

Obviously, there is a defect in the program (maybe you've spotted it already?)

(a) When is a statement B *data dependent* on a statement A? (1p)

(b) When is a statement B *control dependent* on a statement A? (1p)

(c) When calling the method on the array `[1,2,3]`, which program statements is line 10 backward dependent on? (3p)

(d) Repair the method and correct any defects. Clearly state where changes have been made. (2p)

Assignment 4 (Formal Specification: Logic) (6p)

(a) Consider the following propositional logic formula, where p and q are Boolean variables: (2p)

$$(p \wedge q) \wedge (\neg p \vee q)$$

Is the above formula *satisfiable*? Is the above formula *valid*? Show and explain why?

(b) Define the *pre* and *post* conditions for the following `linearSearch` method formally. Explain all predicates/functions that you use in your specification. (4p)
Informally, the `linearSearch` method should take a sorted array and search for the given number in the array. It should return -1 if the given number is not present in the array, and otherwise return an index such that the number is at that place in the array.

```
method linearSearch( a : array<int>, element : int)
    returns (index : int)
      {
              .....
              .....
      }
```

Assignment 5 Formal Specification (9p)

For this assignment, consider a flight ticket booking system (we only consider a **simplified version** of a particular class). Someone has modeled a `FlightBooking` class as:

```
class FlightBooking{
        var name : string;
        var passportNum : string;
        var flightID: int;
        var ticketCode : int;

        predicate ticketCodeValid()
        requires name != null
        requires passportNum != null
        requires flightID != null
        {
        ticketCode == generateCode(name, passportNum, flightID)
        }

        constructor (na : string, pn : string, fl: int)
        requires na != null && pn != null
        ensures ticketCodeValid()
        {
                name := na;
                passportNum := pn;
                ticketCode := generateCode(na, pn, fl);
        }
}

function method generateCode(na: string, pn: string, fl: int)
      : int
requires na != null && pn != null  && fl!= null
{ ... }
```

Bookings for a particular flight is modeled as a simple array of `FlightBooking`'s.

We now want a predicate that checks if an array of `FlightBooking`'s, that denote bookings for a particular flight journey is valid.

Bookings for a particular flight is a non-null array of `FlightBooking` where each element in the array is non-null, and the `flightID` corresponding to each element in the array is equal to the id of the flight, and and the ticket code corresponding to each booking is valid: the stored `ticketCode` is equal to the ticket code that is generated from the other details.

(a) Write down the body of the (4p)

```
predicate validateBookings(bookings :arr<FlightBooking>,
flightID: int)
{ ...  }
```

Use Dafny syntax in your answer. We do not subtract points for minor syntactical errors.

We now want to specify a method with the following type:

```
method checkin(bookingsFl :arr<FlightBooking>,  passportNum :srting,
        tktCode: int, flightID: int) returns (checkinOK : bool)
   requires ?
   ensures ?
```

Informally, the `checkin` method takes a valid set of flight bookings corresponding to that flight, a non-null passport number, ticket code and flight id. It returns true if there is a booking corresponding to the given passport number, and the supplied ticket code is correct, and false in all other cases.

(b) Write down the formal specification of `checkin`. In other words, fill in the `requires` and `ensures` clauses above. Use Dafny syntax in your answer. We do not subtract points for minor syntactical errors. (5p)

Assignment 6 (Formal Verification) (10p)

Consider the following Dafny program:

```
method AlwaysOdd(x : int) returns (y : int)
ensures y%2 == 1;
{
        if (x%2 == 1)
        { y := x+1; }
        else
        { y := x+2;}
        y := (2*y)+1;
}
```

Next, suppose we want to run the following snippet of Dafny code:

```
method Test(){
        var m := AlwaysOdd(2);
        var n := AlwaysOdd(3);
        assert m == n;
        }
```

(a) The above code will cause a Dafny compiler error: (2p)

```
Error:  assertion violation
```

Explain why.

(b) Fix `AlwaysOdd` so that Dafny would be able to prove the assertion. (3p)

(c) Prove that your revised version of `AlwaysOdd` satisfies its post-condition using the weakest pre-condition calculus. Show all details of your proof and motivate each step. (5p)

Weakest pre-condition rules are provided in Page 12.

Assignment 7 (Formal Verification (proving loops)) (5p)

Consider the following Dafny program:

```
method m1(n : nat) returns (i : nat)
requires n >= 0
ensures i == 2*n

{
        i := 0;
        while (i < n)
        invariant i <= n
        decreases n-i
        { i := i + 1; }
        i := 2*i;
}
```

Prove total correctness (including termination) for the above program using the weakest pre-condition calculus (Weakest pre-condition rules are provided in Page 12).

(total 60p)

Additional Notes

Weakest pre-condition rules:

Assignment:	$wp(x := e, R) = R[x \mapsto e]$
Sequential:	$wp(S1; S2, R) = wp(S1, wp(S2, R))$
Assertion:	$wp(\textit{assert } B,\ R) = B\ \&\&\ R$
If-statement:	$wp(\textit{if } B \textit{ then } S1 \textit{ else } S2,\ R) =$ $(B \text{ ==> } wp(S1, R)) \wedge (!B \text{ ==> } wp(S2, R))$
If-statement (empty *else* branch):	$wp(\textit{if } B \textit{ then } S1,\ R) =$ $(B \rightarrow wp(S1, R))\&\&(!B \text{ ==> } R)$
While:	$wp(\textit{while } B\ I\ D\ S,\ R) =$ I $\wedge (B\ \&\&\ I \text{ ==> } wp(S, I))$ $\wedge (!B\ \&\&\ I \text{ ==> } R)$ $\wedge (I \text{ ==> } D >= 0)$ $\wedge (B\ \&\&\ I \text{ ==> } wp(tmp := D; S, tmp > D))$