

Testing, Debugging, and Verification exam
DIT082/TDA567

Day: 9 January 2016

Time: 14⁰⁰ – 18⁰⁰

Responsible:	Atze van der Ploeg
Results:	Will be published mid February or earlier
Extra aid:	Only dictionaries may be used. Other aids are <i>not</i> allowed!
Grade intervals:	U : 0 – 18p, 3 : 19 – 24 p, 4 : 25 – 29p, 5 : 30 – 37p, G : 19 – 29p, VG : 30 – 37p, Max. 37p.

Please observe the following:

- This exam has 8 numbered pages.
Please check immediately that your copy is complete
- Answers must be given in English
- Please use page numbering on your pages
- Please write clearly
- Fewer points are given for unnecessarily complicated solutions
- Indicate clearly when you make assumptions that are not given in the assignment
- Answers to the exam will be published on the course website tomorrow.

Good luck!

1 Testing

Assignment 1 Certainty and Testing

(2p)

In most cases, unit testing can give some assurances, but not guarantees for all inputs.

→ Explain why in most cases unit testing cannot give such hard guarantees.

Assignment 2 Coverage

(4p)

Consider the following piece of Java code:

```
class Group{
    private final String[] names;

    Group(String[] names){
        this.names = names;
    }

    // requires: All elements of names are non-null
    // ensures: returns true if and only if
    //           there is an element in
    //           names that equals name
    boolean isPartOfGroup(String name){
        if(name == null) { return false; }
        for(int i = 0; i < names.length ; i ++){
            if(name.equals(names[i])){
                return true;
            }
        }
        return false;
    }
}
```

→ Construct a Java class where the methods are tests of the `isPartOfGroup` method above, such that the test-cases together provide *statement coverage*.

Assignment 3 Mutation testing

(4p)

Consider the following Java method:

```
/*
requires: input left and right are non-null arrays which are sorted
in non-decreasing order
ensures: output is a non-null array, sorted in non-decreasing order,
such that for any integer i, the number of occurrences in the output
of i, is equal to the number of occurrences in the left arrays of i
plus the number of occurrences in the right array of i. */
public static int[] merge(int[] left, int[] right){
    int [] res = new int[left.length + right.length];
    int il = 0, ir = 0, i = 0;
    while(il < left.length && ir < right.length){
        if(left[il] <= right[ir]){
            res[i] = left[il];
            il += 1; i += 1;
        } else {
            res[i] = right[ir];
            ir += 1; i += 1;
        }
    }
    while (il < left.length) {
        res[i] = left[il];
        il += 1; i += 1;
    }
    while (ir < right.length) {
        res[i] = right[ir];
        ir += 1; i += 1;
    }
    return res;
}
```

Ludvig has constructed a set of tests for this method which consists of the following tests (in shorthand):

```
merge({}, {}) == {}
merge({2,2,3},{1,1,1}) == {1,1,1,2,2,3}
merge({0,1,3,5},{2,4}) == {0,1,2,3,4,5}
```

Ludvig thinks that he does not need more tests: he cannot imagine a bug that he has not tested for. You, as a fresh expert on testing, do not agree with Ludvig.

→ Show that Ludvig is wrong: construct a mutant of the method that does not conform to the specification, but that is not killed by Ludvig's test set.

Assignment 4 Test driven development (2p)

The test driven development methodology is often summarized as *red-green-refactor*.

→ Explain what *red-green-refactor* means.

Assignment 5 Minimization (5p)

Suppose we have method `f` which takes an array of characters as input and suppose that this method computes the output incorrectly if the input contains an even number of 'X' characters (but not zero), and otherwise computes the result correctly.

The shortest example of a string which contains an even number of 'X' characters is the string "XX". However, Sven has used a correct implementation of the `ddMin` algorithm to minimize a failing example of the method `f`, and the result was not "XX" but "XXXX".

- (a) Explain why this is possible. (2p)
- (b) Simulate a run of the `ddMin` algorithm and compute a 1-minimal failing input from the following initial failing input: `[x,a,x,x,c,x,x,x]`. Clearly state what happens at *each step* of the algorithm and what the final result is. (3p)

Assignment 6 Formal Specification (1)

(3p)

CompCert is a *verified* compiler from C to assembly.

- Briefly explain what we mean when we say that CompCert is a verified compiler from C to assembly. Use at least the following words in your answer: specification, behavior, proof.

Assignment 7 Formal Specification (2)

(7p)

In this question you are going to specify and implement a method that gives a *reversed* copy of an array in Dafny. For example, the result of running the method on an array containing [1,2,3,4] will be a new array containing [4,3,2,1]. The header of the method is as follows:

```
method reverse(a : array<int>) returns (res : array<int>)  
requires a != null  
ensures ?
```

- (a) Complete the specification of `reverse` by filling in the `ensures` field. (3p)
- (b) Implement the `reverse` method. Use a `while` loop and provide a loop invariant and decrease clauses such that Dafny will be able to prove total correctness. (It is not allowed to use a `parallel for` loop.) (4p)

Assignment 8 (Formal Verification)

(10p)

A remarkable fact of numbers is that the sum of the natural numbers 0 till n is $\frac{n(n+1)}{2}$.

In other words (assuming $n \geq 2$):

$$0 + 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

For example, $0 + 1 + 2 + 3 + 4 + 5 = \frac{5(5+1)}{2} = 15$

In this question, you are going to prove that $0 + \dots + n = \frac{n(n+1)}{2}$ is true using the weakest-precondition calculus.

The expression $\frac{n(n+1)}{2}$ is implemented by `sumn`:

```
function sumn(n : int) : int { n * (n + 1) / 2 }
```

The following method implements $0 + 1 + 2 + \dots + n$:

```
method sum(n : nat) returns (s : nat)
ensures s == sumn(n)
{
  i := 0;
  s := 0;
  while i < n
  invariant i <= n && s == sumn(i)
  decreases n - i
  {
    i := i + 1;
    s := s + i;
  }
}
```

→ Prove total correctness (including termination) for the above program.

You can assume:

- `sumn(0) = 0`
- `s == sum(i) ==> s + (i + 1) == sum(i+1)`
 (or `sumn(i) + (i + 1) = sumn(i + 1)`)
 (below I explain why this is true in case you are interested, but this is not needed to make the exam.)

Recall that a method without an `requires` clause is the same as a method with the clause `requires true`.

This is the end of the exam, you do not need to read further to make the exam!

Below I explain why the assumptions above are true in case you are interested:

The assumption $s == \text{sum}(i) ==> s + (i + 1) == \text{sum}(i+1)$

follows from $\text{sumn}(i) + (i + 1) = \text{sumn}(i + 1)$

But why does this hold? Here is a proof:

$$\begin{aligned}\text{sumn}(n) + (n + 1) &= \frac{n(n + 1)}{2} + (n + 1) = \frac{n(n + 1) + 2(n + 1)}{2} \\ &= \frac{(n + 2)(n + 1)}{2} = \text{sumn}(n + 1)\end{aligned}$$