Testing, Debugging, and Verification exam DIT082/TDA567

Day: 11 January 2016 Time: $14^{00} - 18^{00}$

Responsible:	Atze van der Ploeg
Results:	Will be published mid February or earlier
Extra aid:	Only dictionaries may be used. Other aids are *not* allowed!
Grade intervals:	**U**: 0 – 21p, **3**: 22 – 32p, **4**: 32 – 40p, **5**: 40 –46p, **G**: 22 – 39p, **VG**: 40 – 46p, **Max.** 46p.

Please observe the following:

- This exam has 10 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 Levels of testing (3p)

Crappy software Inc. has a launced a new social network. A few hours after launch, the users start complaining that there are messages being sent in their name, but that they did not send these messages themselves. After inspection, it turns out that a function `authenticate` authenticates anyone if they simply simply fill in more than 50 characters in the password field.

→ What is lowest level of testing detail at which a test could have caught this bug? Briefly motivate your answer.

Assignment 2 Logic coverage (5p)

Consider the following piece of java code:

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

(a) Construct a minimal set of test-cases for the code snippet above, which satisfy *condition decision coverage.* (3p)

(b) It is *impossible* to construct a set of test-cases for the code snippet above which satisfy *Modified Condition Decision Coverage*(MCDC). Explain why this is the case. (2p)

Assignment 3 Branch coverage (4p)

Consider the following Java method:

```
/* merges two sorted lists

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 occurances in the
         output of i, is equal to the number of occurances in the left
         arrays of i plus the number of occurances 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;
}
```

→ Write down one or more test cases, such that this/these test case(s) together satisfy *branch coverage*. State clearly which parts of the test(s) cover which part of the code.

Assignment 4 Property based testing (3p)

A class for handling dates in a software project provides two methods:

```
// computes the year, when given the number of days since 1 Jan, 1980
// requires : days >= 0
// ensures  : output >= 1980
public static int yearFromDay(int days) ...

// computes the number of days since 1 Jan, 1980, on the first day
// of the given year
// requires : year >= 1980
// ensures  : output >= 0
public static int dayFromYear(int year)
```

You are in charge of testing these methods and you want to use randomized (property based) testing.

→ Which property would you test? Describe the property which should hold if `yearFromDay` and `dayFromYear` are implemented correctly.

Assignment 5 Minimization using DDMin (6p)

(a) The `ddMin` algorithm computes a *1-minimal* failing input. Explain what a *1-minimal* failing input is. (2p)

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 the substring `"foo"` (and otherwise computes the result correctly).

(b) Simulate a run of the `ddMin` algorithm and compute a 1-minimal failing input from the following initial failing input: `[b,a,l,f,o,o,b,a]`. Clearly state what happens at *each step* of the algorithm and what the final result is. (4p)

Assignment 6 Stateful property based-testing (7p)

A *multiset* (or *bag*) is a generalization of the concept of a set that, unlike a set, allows multiple instances of the multiset's elements. The *multiplicity* of an element is the number of instances of the element in a specific multiset.

For example, in the multiset $\{a, a, b\}$, a has multiplicity 2, and b has multiplicity 1.

Suppose we have mutable multiset of integers class in Java:

```
class MultiSet {

   // creates a new empty multiset
   MultiSet() { .. }

   // add an occurance of x to the multiset
   void add(int x) { ... }

   // remove an occurance of to the multiset
   void remove(int x) { ... }

   // gives the multiplicity of x
   int multiplicity(int x)

   // Gives an array, sorted in non-decreasing order, with all elements
   // in the multiset. Each element occurs the same number of times
   // in the array as it's multiplicity in the multiset.
   int[] allElements()
}
```

For example, consider the following method:

```
int[] test(){
  MultiSet x = new MultiSet();
  x.add(2);
  x.add(2);
  x.remove(3);
  x.add(1);
  return x.allElements();
}
```

This will return the array `[1,2,2]`.

Recall from the lectures that an algebraic property of a stateful object is two different methods operating on such a stateful object, such that there never is an (observable) difference between executing one method or executing other method. For example for a *mutable set of integers* a property is:

```
void f(IntSet s, int y) {
  s.add(y);                     void f(IntSet s, int y) {
  s.add(y);                 ==    s.add(y)
}                               }
```

(a) Write down two algebraic properties of the stateful *multiset*. (4p)

(b) Describe how you would use random testing to test this algebraic property. Include the following words in your answer: random, shared prefix, shared postfix, observe state. Don't forget to describe when the test succeeds or fails! (3p)

Assignment 7 Formal Specification (8p)

For a user-authentication mechanism, someone has modeled a user as the following Dafny class:

```
class User{
  var userName : string;
  var password : string;
  var passwordHash : int;

  predicate hashIsCorrect()
  requires password != null
  {
    passwordHash == hashPassWord(password)
  }

  constructor (un : string, pw : string)
  requires un != null && pw != null
  ensures hashIsCorrect()
  {
    userName := un;
    password := pw;
    passwordHash := hashPassword(pw);
  }
}

function hashPashWord(x : string) : int
requires x != null
{ ... }
```

A user database is modelled as simply an array of users. We now want a predicate that checks if an array of users is a valid user database. A valid user database is defined as a non-null array of users where each element is non-null and the hash of each user is correct: the stored password hash is equal to the hash of the password.

(a) Write down the body of the (4p)
`predicate isValidUserDB(users :  arr<User>) { ...  }`
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 authenticate(users : arr<User>,
                    userName : string,
                    passwordHash : int) returns (authenticated : bool)
requires ?
ensures ?
```

Informally, the `authenticate` method takes a valid set of users, a non-null username and a password hash and returns true if the user is in the user database and the supplied password hash is correct, and false in all other cases.

(b) Write down the formal specifation of `authenticate`. 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. (4p)

Assignment 8 (Formal Verification) (10p)

The nth power of 2, 2^n, can be defined in Dafny as follows:

```
function pow2(n : int) : int
requires n >= 0
{
  if n == 0 then 1 else 2 * pow2(n - 1)
}
```

When given a number $x \geq 1$, the following Dafny program computes the number n such that $2^n \leq x < 2^{n+1}$.

```
method log2(x : int) returns (n : int, p : int)
requires x >= 1
ensures  n >= 0 && p == pow2(n) && 0 < p <= x < p * 2
{
  n := 0;
  p := 1;

  while (p * 2 <= x)
  invariant n >= 0 && p == pow2(n) && 0 < p <= x
  {
    n := n + 1;
    p := p * 2;
  }
}
```

(a) Prove partial correctness (no termination proof) for the above program. You can assume that `p == pow2(n) ==> p * 2 == pow2(n+1)`. (5p)

(b) What is a suitable variant (decreases clause) for the while loop in the above program? (1p)

(c) Prove termination of the while-loop for the above program using the variant from the previous sub-question. (4p)

(total 46p)