Testing, Debugging, and Verification re-exam
DIT082/TDA567

Day: 5 April 2016 Time: $14^{00} - 18^{00}$

Responsible:	Atze van der Ploeg
Results:	Will be published mid May 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 14 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 Testing debugging and verification (3p)

Professor Brainy McSmartypants thinks that all software should be fully verified. "Why would anyone not verify their software, certainty is of the utmost import!", he argues.

→ Give a reason why a company would not verify its software but rely on testing instead.

Solution

More certainty costs more effort. It is hence a cost-benefit trade-off wether verification is worth the effort. It is currently highly unlikely that verification is currently worth the effort for programs where certainty is not that important, such as websites and computer games.

Assignment 2 Logic coverage (3p)

Consider the following piece of java code:

```
if (x < 1 || (y > z && z == 3) )
   return x;
else
   return z;
```

→ Construct a minimal set of test-cases for the code snippet above, which satisfy *Modified Condition decision coverage*.

Solution

{x = 2 , y = 1, z = 3} {x = 0 , y = 1 , z = 3} {x = 2 , y = 4 , z = 3} {x = 2 , y = 4 , z = 0}

Assignment 3 Branch coverage (5p)

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 the number of elements that are present in
         both arrays
*/
public static int inBoth(int[] left, int[] right){
  int il = 0, ir = 0, res = 0;
  while(il < left.length && ir < right.length){
    if(left[il] == right[ir]) {
       il += 1; ir += 1; res += 1;
    } else if(left[il] < right[ir]) {
       il += 1;
    } else {
      ir += 1;
    }
  {
  return res;
}
```

(a) Explain why a test set for this program that has statement coverage must also have branch coverage. (2p)

Solution
Statement and branch coverage only differ if there are `if` statements without an `else` clause, which there are not.

(b) 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. (3p)

Solution
For instance we can take the test case : `merge({1,1,3},{1,2}) == 1}`. The first pair `(1,1)` covers the first if, the second pair `(1,2)` covers the second if, and the third pair `(3,2)` covers the third if.

Assignment 4 Property based testing (3p)

A fast way to see if a sorted list contains a certain element is binary search, but its implementation is a bit tricky.

→ How would you use randomized testing of the pointwise equivalence of functions to get some certainty about an implementation of binary search?

Solution

Pointwise equivalence of functions means seeing if two methods compute the same result. Hence, we also implement a linear search method, which should give exactly the same result as the binary search. We then generate random (sorted) lists and elements to check, and see if the result of the linear search and the binary search are the same.

Assignment 5 Minimization using DDMin (7p)

An method for computing the checksum of a string fails if there are two identical characters in a string, for example in `"aa"` or `"ada"`.

(a) List *all* 1-minimal failing subsequences in the following string: `[f,a,e,c,c,a,e,g]`. (2p)

Solution

`"aa","cc","ee"`

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

Solution

(The original exam stated "[f,a,e,x,c,c,a,e,g]", which was a typo. Full point were given for solving "f,a,e,x,c,c,a,e,g" with any way of splitting non-even length strings.)
Start with granularity $n = 2$ and sequence `[f,a,e,c,c,a,e,g]`.

The number of chunks is 2
==> $\texttt{n} : 2, [\texttt{f}, \texttt{a}, \texttt{e}, \texttt{c}]$ PASS (take away first chunk)
==> $\texttt{n} : 2, [\texttt{c}, \texttt{a}, \texttt{e}, \texttt{g}]$ PASS (take away second chunk)

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

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

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

Increase number of chunks to $min(n * 2,\ len([\texttt{l}, \texttt{f}, \texttt{o}, \texttt{o}]) = 4$
==> $\texttt{n} : 4, [\texttt{c}, \texttt{e}, \texttt{g}]$ PASS (take away first chunk)
==> $\texttt{n} : 4, [\texttt{e}, \texttt{e}, \texttt{g}]$ FAIL (take away second chunk)

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

Adjust number of chunks to $max(n-1, 2) = 2$

==> $\texttt{n} : 2, [\texttt{e}]$ PASS (take away first chunk)
==> $\texttt{n} : 2, [\texttt{e}]$ PASS (take away second chunk)

As $n == len([\texttt{e}, \texttt{e}])$ the algorithm terminates with 1-minimal failing input $[\texttt{e}, \texttt{e}]$

Assignment 6 Stateful property based-testing (6p)

Sven has has implemented a stateful set with the following interface:

```
class SvenSet {

   SvenSet() ...

   void add(int x) ...

   void remove(int x) ...

   boolean contains(int x) ...
}
```

(a) Write down the specification of the methods `add` and `remove`. The specifications should be such that the behavior can only that what one would expect from a mutable *set*. (2p)

Solution

For example:

```
void add(int x) ...
requires : nothing
ensures : result.contains(x)

void remove(int x) ...
requires : nothing
ensures : !result.contains(x)
}
```

Sven has implemented the mutable set as follows:

```
class SvenSet {

   ListInteger elems;
   SvenSet() {
     elems = new LinkedListInteger();
   }

   void add(int x) {
     elems.add(x);
   }

   void remove(int x) {
     int i = elems.indexOf(x);
     if( i >= 0) {
        elems.remove(i);
     }
   }

   boolean contains(int x) {
     return elems.indexOf(x) >= 0;
   }
}
```

The documentation of the used methods from `ListInteger` are as follows:

```
public void add(int element)
//Appends the specified element to the end of this list.

public void remove(int index)
// Removes the element at the specified position in this list.

public int indexOf(int element)
// Returns the index of the first occurrence of the specified element
// in this list, or -1 if this list does not contain the element.
```

However, `SvenSet` does not work as one would expect from a *set*.

(b) Describe what is wrong with the implementation. (1p)

Solution

The problem is that the set allows duplicate elements.

(c) Give an example of an *algebraic property* of mutable sets that does not hold for the implementation of SvenSet, but should for mutable sets. In other words give an example of an *algebraic property* with which randomized stateful testing could have found the incorrect behavior of `SvenSet`. (3p)

Solution
For example:

```
void method4a(SvenSet x, int a) {
  x.add(a);
  x.add(a)
  x.remove(a);
}
==
void method4b(MultiSet x, int a) {
}
```

Assignment 7 Formal Specification (8p)

We want to specify the following method in Dafny:

```
method binarySearch( a : array<int>, element : int)
         returns (index : int)
requires sorted(a)
ensures ?
```

Which, informally takes a sorted array and searches for the given number in the array. It returns -1 if the given number is not present in the array, and otherwise returns an index such that the number is at that place in the array.

(a) Make the above informal description formal by filling in the `ensures` clause above. You can assume that `sorted` is defined correctly. Use Dafny syntax. (4p)

Solution

```
   (index == -1 && forall i :: 0 <= i < a.Length ==> a[i] != e)
|| (0 <= index < a.Length && a[index] == e)
```

The sorted predicate is partially defined as follows:

```
predicate sorted(a : array<int>)
reads a
{  ?  }
```

Sorted here means "non-decreasing": elements at bigger indices are never smaller than elements at smaller indices.

(b) Write down the definition of the body of the predicate `sorted`. Use Dafny syntax. (4p)

Solution

```
  a != null && forall i :: 0 <= i < a.Length ==> forall j :: i <= j < a
.Length ==> a[i] <= a[j]
```

Assignment 8 (Formal Verification) (11p)

The *Fibonacci* numbers, $1, 1, 2, 3, 5, 8, 13, ..$ are defined as follows in Dafny:

```
function fib(n : int) : int
{ if n <= 1 then 1 else fib(n-1) + fib(n-2) }
```

Examples:

```
fib(0) == 1, fib(1) == 1, fib(3) == 3
```

The following method computes the nth Fibonacci number, for $n \geq 1$:

```
method fibfast(n: int) returns (r : int)
requires n >= 1
ensures r == fib(n)
{
  var i := 1;
  var p := 1;
  r := 1;
  while(i < n)
  invariant ?
  {
    var tmp := r;
    r := r + p;
    p := tmp;
    i := i + 1;
  }
}
```

The variable p always contains the previous fibonacci number, and r the current.

(a) Give a suitable loop invariant (i.e. a loop invariant such that the post-condition is provable). (2p)

Solution

```
  invariant i <= n && r == fib(i) && p == fib(i-1)
```

(b) Prove partial correctness (no termination proof) for `fibfast` using the loop invariant from the previous sub-question. You may compute `fib` in your answer (for example replace `fib(1)` by `1`). You may also assume that `p == fib(i-1) && r == fib(i) ==> p + r == fib(i + 1)` (5p)

Solution

1) Loop invariant holds on entry:

```
P ==> wp(S, I)
```

Which expands to:

```
i < n ==> wp(i := 1; p := 1; r := 1,
i <= n && r == fib(i) && p == fib(i-1))
```

Compute weakest precondition:

```
wp(i := 1; p := 1; r := 1, i <= n && r == fib(i) && p == fib(i-1))
```

Apply the Seq-rule:

```
wp(i := 1, wp(p := 1; r := 1, i <= n && r == fib(i) && p == fib(i-1)))
```

Apply the Seq-rule:

```
wp(i := 1, wp(p := 1, wp (r := 1,
i <= n && r == fib(i) && p == fib(i-1))))
```

Apply Assignment-rule:

```
wp(p := 1, wp (r := 1, 1 <= n && r == fib(1) && p == fib(1-1)))
```

Apply Assignment rule (2x):

```
1 <= n && 1 == fib(1) && 1 == fib(1-1)
```

Compute: `1 <= n && 1 == 1 && 1 == 1`

Simplify :

```
1 <= n && true && true
```

Simplify :

```
1 <= n
```

Plug in weakest precondition:

```
n >= 1 ==> 1 <= n
```

Which is true by elemental algebra.

2) Prove loop invariant

```
E && I ==> wp(A,I)
```

Which expands to:

```
i < n && i <= n && r == fib(i) && p == fib(i-1) ==>
wp(tmp := r;r := r + p;p := tmp;i := i + 1,
i <= n && r == fib(i) && p == fib(i-1))
```

Compute weakest precondition:

```
wp(tmp := r;r := r + p;p := tmp;i := i + 1,
i <= n && r == fib(i) && p == fib(i-1))
```

Apply the Seq-rule(3x):

```
wp(tmp := r, wp (r := r + p, wp (p := tmp, wp (i := i + 1,
i <= n && r == fib(i) && p == fib(i-1)))))
```

Apply Assignment(3x):

```
i <= n && r + p == fib(i + 1) && r == fib(i)
```

Plug in weakest precondition:

```
i < n && i <= n && r == fib(i) && p == fib(i-1) ==>
i <= n && r + p == fib(i + 1) && r == fib(i)
```

Since `i < n` implies `i <= n`

```
r == fib(i) && p == fib(i-1) ==> r + p == fib(i + 1)
```

True by `p == fib(i-1) && r == fib(i) ==> p + r == fib(i + 1)` (given).

3) Loop invariant implies post condition

```
I && !E ==> wp(C, Q)
```

Which expands to:

```
i <= n && r == fib(i) && p == fib(i-1) && !(i < n) ==> wp( ,r == fib(n))
```

By empty rule:

```
i <= n && r == fib(i) && p == fib(i-1) && !(i < n) ==> r == fib(n)
```

Simplify:

```
i <= n && r == fib(i) && p == fib(i-1) && i >= n ==> r == fib(n)
```

Since `i <= n && i >= n ==> i == n`, replace `i` by n.

```
n <= n && r == fib(n) && p == fib(n-1) && n >= n ==> r == fib(n)
```

Which is true by `p ==> p == True`.

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

Solution

```
n - i
```

(d) Prove termination of the while-loop for the above program using the variant from the previous sub-question. (3p)

Solution

1) Decreases clause is always $\geq$ 0.

```
I ==> D >= 0
```

Which expands to:

```
i <= n && r + p == fib(i + 1) && r == fib(i)&& ==> n - i >= 0
```

Remove irrelavant bits:

```
i <= n ==> n - i >= 0
```

Rewrite.

```
i <= n ==> n >= i
```

True by elemental algebra.

2) Decreases clause decreases each iteration.

```
E && I ==> wp(tmp := D ; S, tmp > D)
```

Which expands to:

```
i < n && i <= n && r + p == fib(i + 1) && r == fib(i) ==>
wp(tmp2 := n - i; tmp := r;r := r + p;p := tmp;i := i + 1, tmp2 > n - i)
```

Compute weakest precondition:

```
wp(tmp2 := n - i; tmp := r;r := r + p;p := tmp;i := i + 1, tmp2 > n - i)
```

By Seq and assignment rule rule (4x):

```
 n - i > n - (i + 1)
```

Simplify

```
n - i > n - i - 1
```

Simplify

```
0 > - 1
```

(total 46p)