ETH zürich

Alexander J. Summers

Program Verification

Exercise Solutions 8: The Boogie Intermediate Verification Language

Assignment 1 (Modularity)

The following program satisfies all of the requirements:

```
var x : int; // global variable

procedure p()
  ensures x > 0; // implicitly: requires true;
  modifies x;
{
  x := x + 1;
}

procedure test()
  modifies x;
{
  x := 0;
  call p();
  assert x == 1; // for the second part of the question
}
```

In particular, running this program through Boogie will yield two verification errors: one for the postcondition of `p` and one for the `assert` statement.

Assignment 2 (Verification)

```
const length : int;

var m:[int]int;
var n:[int]int;

procedure n_reverse_m()
  modifies n;
  ensures (forall i:int :: {n[i]} // good choice of trigger
    0<=i && i<length ==>
      n[i] == m[length - i - 1]);
{
  var i : int;
  i := 0;
  while(i < length)
    invariant (forall j:int :: {n[j]} 0 <= j && j < i ==>
                 n[j] == m[length - j - 1]);
    {
      n[i] := m[length - i - 1];
      i := i + 1;
    }
}
```

Assignment 3 (Well-Definedness Conditions)

1. $\mathrm{def}(\texttt{e[e']}) = \mathrm{def}(\texttt{e})$ && $\mathrm{def}(\texttt{e'})$ && 0 <= e' && e' < length(e)
2. $\mathrm{def}(\texttt{e}_1 \texttt{ == e}_2) = \mathrm{def}(\texttt{e}_1)$ && $\mathrm{def}(\texttt{e}_2)$
3. $\mathrm{def}(\texttt{!e}) = \mathrm{def}(\texttt{e})$
4. $\mathrm{def}(\texttt{e}_1 \texttt{ || e}_2) = \mathrm{def}(\texttt{e}_1)$ && (!$\texttt{e}_1$ ==> $\mathrm{def}(\texttt{e}_2)$)
   Note: this definition accounts for a short-circuiting semantics of disjunctions.

Assignment 4 (Boogie Maps and Java Arrays)

Encoding in Boogie based on the arrays encoding from slide 197:

```
type IntArray;
function addr(a:IntArray) : int;
function length(a:IntArray) : int;

var Heap: [int]int; // entire memory

procedure test(a: IntArray)
  modifies Heap;
{
  var b: IntArray;
  b := a;
  Heap[addr(b) + 3] := 4;
  assert Heap[addr(a) + 3] == 4;
}
```

The Boogie maps are immutable values. Therefore, the assignment `b := a` in Boogie would assign a **copy** of the map `a` to `b`. Similarly, `b[3] := 4` is just a syntactic sugar for `b := b[3:=4]` where `b[3 := 4]` is a **new** map that is identical to the original map `b` except that `3` is mapped to `4`. As a result, after executing the following Boogie statements:

```
b := a;
b[3] := 4;
```

we will not be able to assert that `a[3] == 4`. Instead, we will be able to assert that `a[3]` has still the same value it had before executing these two statements.

On the other hand, the assignment `b = a` in Java makes `b` to point to the same memory block to which `a` points. Because of this **aliasing**, changes via variable `b` can be observed via variable `a`.