hw1 weakest precondition calculus and loops

James Fitzgerald

Question 1

Post condition

$Q=(0\leq y)\wedge(0\leq x\Rightarrow y=x)\wedge(x<0\Rightarrow y=-x)$

If else B=(x<0)

$WP(\text{if } B \text{ then } S1 \text{ else } S2,Q)=(B\Rightarrow WP(S1,Q))\wedge(\neg B\Rightarrow WP(S2,Q))$

Substitute

$((x<0)\Rightarrow(0\leq -x)\wedge(0\leq x\Rightarrow -x=x)\wedge(x<0\Rightarrow -x=-x))$

$\wedge((0\leq x)\Rightarrow(0\leq x)\wedge(0\leq x\Rightarrow x=x)\wedge(x<0\Rightarrow x=-x))$

Simplify

$((x<0)\Rightarrow(x\leq 0)\wedge(0\leq x\Rightarrow -x=x)\wedge(x<0\Rightarrow -x=-x))$

$\wedge((0\leq x)\Rightarrow(0\leq x)\wedge(0\leq x\Rightarrow x=x)\wedge(x<0\Rightarrow x=-x))$

TRUE

Question 2

Part 1

Post condition

$Q= big>small$

If else x>y

$WP(\text{if } B \text{ then } S1 \text{ else } S2,Q)=(B\Rightarrow WP(S1,Q))\wedge(\neg B\Rightarrow WP(S2,Q))$

Substitute {big, small := x, y} in the then and {big, small := y, x;} in the else

$((x>y) \Rightarrow(x>y))$

$\wedge((x\leq y) \Rightarrow(y>x))$

The precondition fails because (x≤y) can be true at the same time that (y>x) is false

So x != y ==> wp(S,Q)

Part 2

```
method Q2(x : int, y : int) returns (big : int, small : int)
  requires x != y
  ensures big > small
{
  if (x > y)
   {big, small := x, y;}
  else
   {big, small := y, x;}
}
```

Question 3

Part A

Post condition

Res = n0 * m0

Loop invariant

I: res + n * m == n0 * m0

```
method Q3(n0 : int, m0 : int) returns (res : int)
ensures res == n0 * m0
{
  var n, m : int;
  res := 0;
  if (n0 >= 0)
    {n,m := n0, m0;}
  else
    {n,m := -n0, -m0;}
  while (0 < n)
    invariant res + n * m == n0 * m0
    invariant n >= 0
  {
    res := res + m;
    n := n - 1;
  }
}
```

Part B

Holds before loop

If res is 0 and n is n0 then 0+(n0*m0) = n0 * m0

Holds during loop

As n decreases res increases by m so for example, Iteration * m0 + (n0-iteration * m0) = n0 * m0

Holds after loop

Once n == 0 then it is just the regular multiplication again

Iteration * m0 + (0-iteration * m0) = n0 * m0

the decrease expression is bounded below by zero

The loop ends if n == 0 and if the n is less than zero it would break the invariant

the decreasing expression decreases on each iteration

n := n - 1

Question 4

Part A

```
method ComputeFact(n : nat) returns (res : nat)
requires n > 0
ensures res == fact(n)
{
  res := 1;
  var i := 2;
  while (i <= n)
    invariant 2 <= i <= n + 1
    invariant res == fact(i-1)
  {
    res := res * i;
    i := i + 1;
  }
}

function fact(n: nat): nat
{
  if n == 0 then 1
  else n * fact(n - 1)
}
```

Part B: Construct a paper-pencil proof of total correctness of the loop.

- *I: invariant holds *before* the loop*
  - i starts out as 2 and n has to be at least 1 so 2<= I <= (at least 1) +1
  - i-1=1 and fact(1) = 1 and res =1 so res== fact(i-1)

- *(forall xs, I /\ E ==> wp(S,I)) : If the invariant holds before the loop then it must hold after the loop body on an iteration*
  - Since i is increasing then it remains more than 2 and since the loop ends when i > n then we know i is less than or equal to n +1
  - res * i = fact(i-2) because each loop it multiplies by itself which is what fact does in reverse with recursion
- *(forall xs, I /\ !E ==> Q): If the loop invariant holds and the loop exits, then it must satisfy the post-condition.*
  - The invariant still holds because it is what the loop means. When I = n+1 it is over
  - res * i = fact(i-2) because each loop it multiplies by itself which is what fact does in reverse with recursion
- *(forall xs, I /\ E ==> D > 0): the decrease expression is bounded below by zero if the loop body is traversed.*
  - Decrease expression of n-i is bounded by zero because as I goes up n-I goes down until it reaches zero
- *(forall xs, I /\ E ==> wp(S, old(D) > D)): the decrease expression decreases on each iteration (*old* is the Dafny keyword).*
  - Each iteration i = i+1 so n-i goes down