More Proofs Over Procedures

Curt Clifton
Rose-Hulman Institute of Technology

Recall the Steps

1. Specify the procedure
2. Prove that procedure satisfies its specification
3. Use the specification to reason about calls

General Rule, part 1

1. TAKE METHOD SPEC:

```
//@ requires P;
//@ assignable s1, ... sm;
//@ ensures Q;
T p(x1,...,xn) {
  S;
  return r;
}
```

2. PROVE IMPL. CORRECT:

```
  //@ assert P && s01 == s1 && ... && s0m == sm;
S;
  //@ assert Q[r/\result, s01/\old(s1),..., s0m/\old(sm)];
```

INCL. NO SIDE EFFECTS EXCEPT TO $s_1, \ldots s_m$

General Rule, part 2

1. TAKE METHOD SPEC:

```
//@ requires P;
//@ assignable s1, ... sm;
//@ ensures Q;
T p(x1,...,xn) {...}
```

2. PROVE IMPLEMENTATION CORRECT…

3. REASON ABOUT CALLS:

```
  //@ assert P[e1/x1,...,en/xn] && R1
  //@                 && s01 == s1 && ... && s0m == sm;
v = p(e1,...,en);
  //@ assert Q [e1/x1,...,en/xn,v/\result,
  //@                  s01/\old(s1),..., s0m/\old(sm)] && R2;
```

WHERE $R_1 \Rightarrow R_2$ AND $v, s_1, \ldots s_m$ NOT FREE IN R_1 AND R_2

Example, part 2

GIVEN:

```
//@ requires acc != 0;
//@ assignable v, x;
//@ ensures v == acc + \old(v)
//@         && x == \old(v) + \old(x);
void accel(double acc) {
      x = v + x;
      v = acc + v;
}
```

FIND WEAKEST PRE-CONDITION:

```
      //@ assert ???
g = -10;
accel(g);
      //@ assert x == 0 && v == 20;
```

Solution

```
9         //@ assert 30 == v && -30 == x;
8         //@ assert 30 == v && 0 == 30 + x;
7         //@ assert true && 30 == v && 0 == v + x;
6         //@ assert -10 != 0 && 20 == -10 + v  && 0 == v  + x;
      g = -10;
5         //@ assert g != 0 && 20 == g + v  && 0 == v  + x;
4         //@ assert g != 0 && v0 == v && x0 == x && 20 == g + v0 && 0 == v0 + x0;
      // ----- original expansion, replaced with surrounding assertions
0         //@ assert g != 0 && v0 == v && x0 == x;
      accel(g);
1         //@ assert v == g + v0 && x == v0 + x0;
      // ----- original expansion, replaced with surrounding assertions
2,3       //@ assert 20 == g + v0 && 0 == v0 + x0 && x == 0 && v == 20;
          //@ assert x == 0 && v == 20;
```

Another Example

ASSUME THIS METHOD SATISFIES ITS SPEC:

```
//@ requires n >= 0;
//@ assignable \nothing;
//@ ensures \result == (\product int k; 1 <= k && k <= n; k));
public int fact(n);
```

FIND THE WEAKEST PRE-CONDITION:

```
      //@ assert ???
g = g * 2;
r = fact(g);
      //@ assert r >= 120;
```

THIS IS Q1

Another Solution

```
7         //@ assert g >= 2.5;
5         //@ assert g >= 0 && g * 2 >= 5;
5         // ==> by definition of factorial (i.e., the mathematical function)
4         //@ assert g * 2 >= 0 && (\product int k; 1 <= k && k <= g * 2; k) >= 120;
      g = g * 2;
0         //@ assert g >= 0
3         //@       && (\product int k; 1 <= k && k <= g; k) >= 120;
      r = fact(g);
1         //@ assert r == (\product int k; 1 <= k && k <= g; k)
2         //@        && (\product int k; 1 <= k && k <= g; k) >= 120;
          //@ assert r >= 120;
```

Another Example

```
void setRange(int a, int b) {
    if (a < b) {
        lo = a;
        hi = b;
    } else {
        lo = b;
        hi = a;
    }
}
```

- What should the specification be?

Yet Another Example

ASSUME THIS METHOD SATISFIES ITS SPEC:

```
//@ requires (\forall int k; 0 <= k && k < a.length; a[k] >= 0);
//@ assignable \nothing;
//@ ensures \result == (\sum int k; 0 <= k && k < a.length; a[k]);
double sumAll(double[] a) { ... }
```

FIND THE SPECIFICATION FOR THE FOLLOWING METHOD AND PROVE IT CORRECT:

```
void setAverageScore() {
    double s = sumAll(scores);
    avg = s / scores.length;
}
```

THIS IS Q2

A Pitfall — Aliasing

- What's aliasing?
- Why is it an issue?
- Consider: What's the post-condition for

  a = sum(a, b);

  using our rules?


```
    //@ assert true;
a = sum(a, b);
    //@ a == a + b;
```

WHAT MUST b BE APPARENTLY?

Rule: No Aliasing

- In a procedure call
  v = **p(**e_1, …, e_n**);**
  neither v nor any location assignable by **p** may appear in e_1, …, e_n
- "For proofs, introduce new variables to avoid dealing with aliasing"

- Replace
  a = sum(a, b);
  with
  int aOld = a;
  a = sum(aOld, b);

Example

FIND WEAKEST PRE-CONDITION:

```
a = sum(a, b);
    //@ assert a > 0;
```

SOLUTION:

```
8         //@ assert a + b > 0
1     int aOld = a;
7         //@ assert aOld + b > 0
2,6       //@ assert true && aOld + b > 0
1     a = sum(aOld, b);
3-5       //@ assert a == aOld + b && a > 0 && aOld + b > 0;
          //@ assert a > 0;
```