

Proofs of Procedures

Curt Clifton

Rose-Hulman Institute of Technology



PHOTO BY DIGITALART2 - [HTTP://FLIC.KR/P/4CJ6BF](http://FLIC.KR/P/4CJ6BF)

A BIG BITE O' ELEPHANT

COMPLEX RULES FOR PROCEDURES → WE'LL BUILD UP TO THE FULL RULES

Method Specifications in JML

Design by Contract

- * A software development methodology
- * Engineer specifies a contract for each method:
 - * A precondition and
 - * A postcondition
- * Programmer implements the methods

JML Syntax: Preconditions

- * Specified with “requires” clauses
 - * **requires** <predicate>;
 - * Example: **requires x > 0;**
- * <**predicate**> is (almost) any boolean-valued Java expression
 - * No side effects; can use special JML constructs.

JML Syntax: Postconditions

- * Specified with “ensures” clauses

- * **ensures <predicate>;**

- * Example:

```
ensures \result * \result <= x &&  
        x < (\result + 1) * (\result + 1);
```

JML EXPRESSION TO REFER TO METHOD'S RETURN VALUE

Reference of Some Other JML Expressions

Expression	Meaning
<code>\result</code>	method's return value
<code>\old(<expr>)</code>	value of <code><expr></code> in "pre-state" of method
<code><pred₁> ==> <pred₂></code>	true if <code>pred₁</code> implies <code>pred₂</code>
<code><pred₁> <==> <pred₂></code>	true if <code>pred₁</code> is logically equivalent to <code>pred₂</code>

BACKSLASHES AVOID CLASHES

Pure as the driven snow

- * Recall that JML annotations can call side-effect-free methods
 - * **//@ requires !isEmpty();**
//@ ensures ...*elided*...;
public Object remove(Object o) ...
- * Need to know that isEmpty() is side-effect-free
 - * **public /*@ pure @*/ boolean isEmpty() ...**

Shades of Purity?

Assignable Clauses

- * Assignable clauses specify “frame conditions”:
 - * **//@ assignable x, v;**
 - * **//@ assignable a[*];**
 - * **//@ assignable a[j], a[lo..hi];**
 - * **//@ assignable \everything;**
 - * **//@ assignable \nothing;**

 **EQUIVALENT TO DECLARING METHOD pure**

Examples

```
//@ requires l <= r && r < a.length;  
//@ assignable a[l..r];  
//@ ensures a[r] == (\max int k; l <= k && k <= r; a[k]);  
public void bubble(int[] a, int l, int r);
```

```
//@ requires true;  
//@ assignable a[*];  
//@ ensures (\forall int k;  
//@         0 <= k && k < a.length;  
//@         a[k] == \old(a[k] + 1));  
void addOne(int[] a) {
```



Basic Pattern

1. TAKE METHOD SPEC:

```
//@ requires P;  
//@ assignable \nothing;  
//@ ensures Q;  
void p() { S; }
```

**MUST ARGUE THAT *S*
HAS NO SIDE EFFECTS**



2. PROVE IMPL. CORRECT:

```
//@ assert P;  
S;  
//@ assert Q;
```

3. REASON ABOUT CALLS:

```
//@ assert P;  
p();  
//@ assert Q;
```

Example

METHOD SPEC.

```
//@ requires true;  
//@ assignable \nothing;  
//@ ensures x >= 0 && y >= 0;  
void checkQuadrant() {  
    while (x < 0 || y < 0) {  
        // infinite loop  
    }  
}
```

PARTIAL CORRECTNESS PROOF

```
//@ assert true;  
while (x < 0 || y < 0) {  
    //@ assert true && (x < 0 || y < 0);  
    // infinite loop  
    //@ assert true;  
}  
  
//@ assert true && x >= 0 && y >= 0;  
//@ assert x >= 0 && y >= 0;
```

NOTE: NO ASSIGNMENTS

REASONING ABOUT A CALL

```
//@ assert true;  
checkQuadrant();  
//@ assert x >= 0 && y >= 0;
```

WHERE DID x AND y COME FROM?

Pure Procedures with Args – Simple Version

ASSUMING ARGUMENTS AT USE MATCH FORMAL PARAMETER NAMES

1. TAKE METHOD SPEC:

```
//@ requires P;  
//@ assignable \nothing;  
//@ ensures Q;  
void p(x1,...,xn) { S; }
```

2. PROVE IMPL. CORRECT:

```
//@ assert P;  
S;  
//@ assert Q;  
INCL. NO SIDE EFFECTS
```

3. REASON ABOUT CALLS:

```
//@ assert P;  
p(x1,...,xn);  
//@ assert Q;
```

Example

METHOD SPEC.

```
//@ requires true;
//@ assignable \nothing;
//@ ensures x >= 0 && y >= 0;
void checkQuadrant(int x, int y) {
    while (x < 0 || y < 0) {
        // infinite loop
    }
}
```

PARTIAL CORRECTNESS PROOF

```
//@ assert true;
while (x < 0 || y < 0) {
    //@ assert true && (x < 0 || y < 0);
    // infinite loop
    //@ assert true;
}

//@ assert true && x >= 0 && y >= 0;
//@ assert x >= 0 && y >= 0;
```

NOTE: NO ASSIGNMENTS

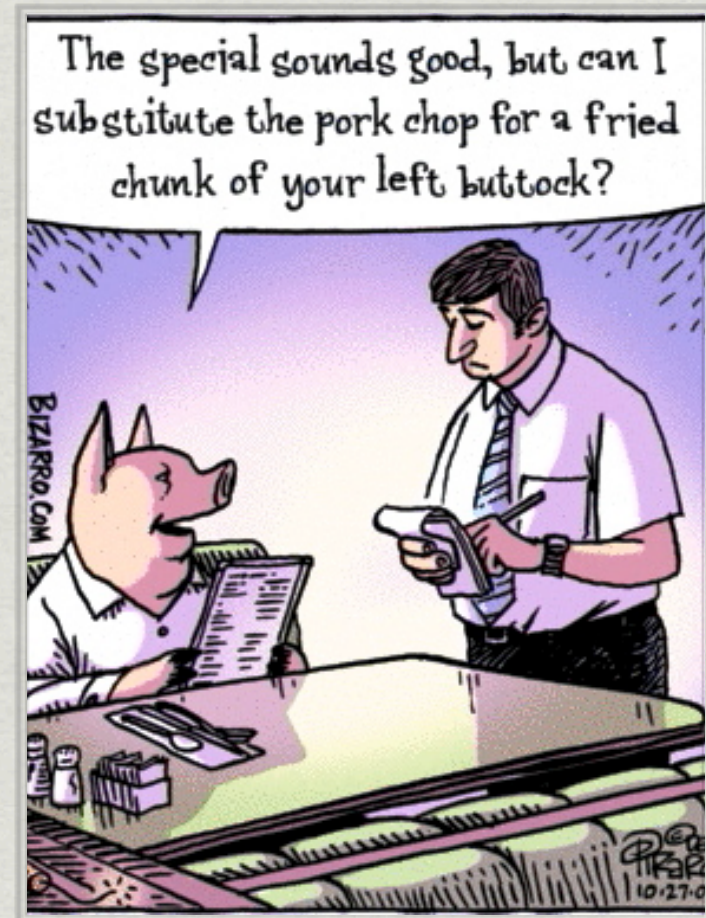
REASONING ABOUT A CALL

```
//@ assert true;
checkQuadrant(x, y);
//@ assert x >= 0 && y >= 0;
```

NOW x AND y CAN COME FROM ANYWHERE

Notation: Substitution

- * $P[x/y]$
 - * Means: substitute x for every (free) occurrence of y in P
- * Example:
 - * P is $0 < y \ \&\& \ y < 8$
 - * $P[x/y]$ is $0 < x \ \&\& \ x < 8$



<http://www.bizarro.com/>

Pure Procedures with Args – Full Version

ALLOWS ARBITRARY ARGUMENTS AND RETURN VALUES

1. TAKE METHOD SPEC:

```
//@ requires P;  
//@ assignable \nothing;  
//@ ensures Q;  
T p(x1, ..., xn) {  
  S;  
  return r;  
}
```

2. PROVE IMPL. CORRECT:

```
//@ assert P;  
S;  
//@ assert Q[r/\result];  
INCL. NO SIDE EFFECTS
```

3. REASON ABOUT CALLS:

```
//@ assert P[e1/x1, ..., en/xn];  
v = p(e1, ..., en);  
//@ assert Q[e1/x1, ..., en/xn, v/\result];
```

Example

GIVEN:

```
//@ requires true;  
//@ assignable \nothing;  
//@ ensures \result == x + y;  
int sum(int x, int y) {  
    int r = x + y;  
    return r;  
}
```

PROVE CORRECT:

```
    //@ assert a == 2;  
b = 3;  
c = sum(a, b);  
    //@ assert c == 5;
```

Answer

GIVEN:

```
//@ requires true;  
//@ assignable \nothing;  
//@ ensures \result == x + y;  
int sum(int x, int y) {  
    int r = x + y;  
    return r;  
}
```



```
1:    //@ assert true;  
3:    //@ assert x + y == x + y;  
int r = x + y;  
2:    //@ assert r == x + y;  
return r;
```

PROVE CORRECT:

```
    //@ assert a == 2;  
b = 3;  
c = sum(a, b);  
    //@ assert c == 5;
```



```
    //@ assert a == 2;  
6    //@ assert a + 3 == 5;  
b = 3;  
5    //@ assert a + b == 5;  
0,4  //@ assert true && a + b == 5;  
c = sum(a, b);  
1-3  //@ assert c == a + b && c == 5  
    //@           && a + b == 5;  
    //@ assert c == 5;
```

That which doesn't change stays the same

1. TAKE METHOD SPEC:

```
//@ requires P;  
//@ assignable \nothing;  
//@ ensures Q;  
T p(x1, ..., xn) {  
  S;  
  return r;  
}
```

2. PROVE IMPL. CORRECT:

```
//@ assert P;  
S;  
//@ assert Q[r/\result];  
INCL. NO SIDE EFFECTS
```

3. REASON ABOUT CALLS:

```
//@ assert P[e1/x1, ..., en/xn] && R1;  
v = p(e1, ..., en);  
//@ assert Q [e1/x1, ..., en/xn, v/\result] && R2;  
WHERE R1 ⇒ R2 AND v NOT FREE IN R1 AND R2
```

Example

FIND THE WEAKEST PRE-CONDITION

```
//@ assert ???
```

```
b = 17;
```

```
c = sum(a, b);
```

```
//@ assert c == 59 && z > 1;
```

Solution

FIND THE WEAKEST PRE-CONDITION

```
    //@ assert ???  
b = 17;  
c = sum(a, b);  
    //@ assert c == 59 && z > 1;
```



```
7    //@ assert a == 42 && z > 1;  
6    //@ assert 59 == 17 + a && z > 1;  
    b = 17;  
5    //@ assert 59 == a + b && z > 1;  
0,4  //@ assert true && 59 == a + b && z > 1;  
    c = sum(a, b);  
1,3  //@ assert c == a + b && 59 == a + b && z > 1;  
2    //@ assert c == a + b && c == 59 && z > 1;  
    //@ assert c == 59 && z > 1;
```

General Rule, part 1

1. TAKE METHOD SPEC:

```
//@ requires P;  
//@ assignable  $s_1, \dots, s_m$ ;  
//@ ensures Q;  
T p( $x_1, \dots, x_n$ ) {  
  S;  
  return r;  
}
```

New spec-only variables introduced to “remember” pre-state values so we can reference them in post-condition.

2. PROVE IMPL. CORRECT:

```
//@ assert P &&  $s_{01} == s_1$  && ... &&  $s_{0m} == s_m$ ;  
S;  
//@ assert Q[r/\result,  $s_{01}/\text{old}(s_1), \dots, s_{0m}/\text{old}(s_m)$ ];
```

INCL. NO SIDE EFFECTS EXCEPT TO s_1, \dots, s_m

Example, part 1

PROVE CORRECT:

```
//@ 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;  
}
```



```
0    //@ assert acc != 0 && v0 == v && x0 == x;  
    //@ assert acc + v == acc + v && v + x == v + x && v0 == v && x0 == x;  
3    //@ assert acc + v == acc + v0 && v + x == v0 + x0;  
    x = v + x;  
2    //@ assert acc + v == acc + v0 && x == v0 + x0;  
    v = acc + v;  
1    //@ assert v == acc + v0 && x == v0 + x0;
```

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, \dots, 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, \dots, s_m NOT FREE IN R_1 AND R_2

Example, part 2

FIND WEAKEST PRE-CONDITION:

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

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;  
}
```

```
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;
```