Recursion, Data Modeling with JML

Curt Clifton
Rose-Hulman Institute of Technology

Recursion

- How do we prove a recursive procedure is correct?
- Assume that it is!
- Really proof by induction
  - Some path through the procedure must not recurse – base case
  - Other paths recurse – induction step

Example

PROVE CORRECTNESS:

```
//@ requires b >= 0;
//@ assignable \nothing;
//@ ensures \result == a * b;
int mult(a, b) {
    int r;
    if (b > 0) {
        r = mult(a, b - 1);
        r = r + a;
    } else {
        r = 0;
    }
    return r;
}
```

```
        //@ assert b >= 0;
int r;
if (b > 0) {
                //@ assert b >= 0 && b > 0;
                //@ assert b >= 1;
                //@ assert b - 1 >= 0;
        r = mult(a, b - 1);
                //@ assert r == a * (b - 1);
                //@ assert r == a * b - a;
                //@ assert r + a == a * b;
        r = r + a;
                //@ assert r == a * b;
} else {
                //@ assert b >= 0 && b <= 0;
                //@ assert b == 0;
                //@ assert 0 == b;
                //@ assert a * b == a * 0;
                //@ assert 0 == a * b;
        r = 0;
                //@ assert r == a * b;
}
        //@ assert r == a * b;
return r;
```

Partial vs. Total

- We only proved **mult** partially correct
- How would we prove it totally correct?
- Two steps
  - Show that arguments to recursive calls change consistently (monotonicity)
  - Show that base case exists

Design By Contract for Objects

DBC for Methods

- Pre-conditions – **requires**
- Post-conditions – **ensures**
- Frame axioms – **assignable**

DBC for Objects

- Besides specifying what methods do, we also have to constrain what the objects can do
- We use ***class invariants*** for this
- Example: `//@ invariant size <= capacity;`

An Example, but first…

We need to configure **ant** to run with the JDK.

- In Eclipse: **Window → Preferences**
  - Select **ant → runtime**
  - Click **Global Entries**
  - **Add External JARs**
  - Find **C:\...JDK…\tools.jar**
    - E.g., C:\Program Files\Java\jdk1.6…\lib\tools.jar

Check your installation

- Right-click **Stack/build.xml**
- **Run As… → Ant Build…**
  - Check the **Hello** target and **Run**

The “…” matters!

Example

- BoundedStack
  - A stack with a fixed maximum size

The Outline

```
public class BoundedStack {
    public BoundedStack(int maxSize) {…}
    public void push(Object x) {…}
    public void pop() {…}
    public Object top() {…}
}
```

1. Specify the Data

LETS US TALK ABOUT size IN OUR SPECIFICATION

```
public class BoundedStack {
    private /*@ spec_public @*/ int size;
    private /*@ spec_public @*/
                        Object[] elems;
    ...
```

2. Specify the Invariant

```
public class BoundedStack {
    private /*@ spec_public @*/ int size;
    private /*@ spec_public @*/ Object[] elems;

    //@ public invariant 0 <= size;
    //@ public invariant elems != null;
    //@ public invariant size <= elems.length;
    //@ public invariant \typeof(elems) == \type(Object[]);
    /*@ public invariant (\forall int i;
      @                        size <= i && i < elems.length;
      @                        elems[i] == null);
      @*/
```

DISALLOWS JAVA WEIRDNESS
LIKE elems = new String[]

3. Specify Public Constructors/Methods

```
/*@ requires n > 0;
  @ assignable elems, size;
  @ ensures elems.length == n && size == 0;
  @*/
public BoundedStack(int n) {
    elems = new Object[n];
    size = 0;
}
```

Don’t Get Pushy

```
/*@ requires size < elems.length;
  @ assignable elems[size], size;
  @ ensures size == \old(size) + 1;
  @ ensures elems[size-1] == x;
  @*/
public void push(Object x) {
    elems[size] = x;
    size = size + 1;
}
```

All Around the Shoemaker's Bench…

```
/*@ requires size > 0;
  @ assignable size, elems[size-1];
  @ ensures size == \old(size-1);
  @*/
public void pop() {
    size = size - 1;
    elems[size] = null;
}
```

REMEMBER THE INVARIANT INCLUDED:

```
(\forall int i; size <= i && i < elems.length; elems[i] == null)
```

I'm on the Top of the World…

```
/*@ requires size > 0;
  @ assignable \nothing;
  @ ensures \result == elems[size-1];
  @*/
public Object top() {
    return elems[size-1];
}
```

Steps for Specifying a Class

- 1. Specify the data…
  - Using **spec_public** fields for now
  - Later we'll see how to be more abstract
- 2. Specify **invariant**s
- 3. Specify each public method/constructor using:
  - **requires**, **assignable**, and **ensures**