Advanced Programming Topics

Proof by induction, and structural induction
Lecture 5
James Marshall

Testing vs Proof

The usual software engineering approach to checking program correctness is *testing.* Test cases are selected, usually according to some strategy such as

- *black-box testing*
- *white-box testing*

Imagine designing black-box test cases for a function

```
fastIntExp :: Integer -> Integer -> Integer
```

that is supposed to raise its first argument to the power of its second, and return the result.

Test case:

Result:

Now look at the code and design some white-box test cases:

```
fastIntExp :: Integer -> Integer -> Integer
fastIntExp a b
    | b == 1    = a
    | even b    = (fastIntExp a (half b))^2
    | otherwise = (fastIntExp a (b-1)) + a
        where half x = truncate ((fromIntegral x)/2)
```

Test case:

Result:

Exercise: fix the `fastIntExp` *function*

Although looking at the code can give us ideas for test cases to pick up particular errors, we still rely on our intuition and experience to generate all the right tests. We may miss some though…

Proof by Induction

To prove that a desired property holds for all *defined* values computed by a function we use *proof by induction*. The major steps are as follows

1. Identify the proof goal
2. Identify the appropriate base case and prove the goal for that case
3. Identify the inductive hypothesis for the inductive step
4. Show that the base case and inductive hypothesis together prove the goal for all defined values of the function

Example - Slow Integer Exponentiation

Prove for all `n > 0` that `slowIntExp a b == a^b`

```
slowIntExp :: Integer -> Integer -> Integer
slowIntExp a 0 = 1                                  --(exp0)
slowIntExp a b = a * (slowIntExp a (b-1))           --(exp+)
```

1. Proof goal is as described above

2. Choose base case `slowIntExp a 0 == a^0`

Prove base case:

3. Identify inductive hypothesis and inductive step

4. Show that (base case $\wedge$ inductive hypothesis) $\Longrightarrow$ proof goal

Structural Induction

We can also use proof by induction to prove properties of data structures, such as lists.

Example - from Thompson (chapter 8)

Prove, given the following functions

```
sum :: [Int] -> Int
sum []     = 0                                        --(sum.1)
sum (x:xs) = x + sum xs                               --(sum.2)

doubleAll []     = []                              --(doubleAll.1)
doubleAll (z:zs) = 2*z : doubleAll zs              --(doubleAll.2)
```

that for all lists `xs`

```
sum (doubleAll xs) = 2 * sum xs                     --(sum+dblAll)
```

1. Proof goal is as just described

2. Prove base case:

3. Identify inductive hypothesis and inductive step

4. Show that (base case $\wedge$ inductive hypothesis) $\Longrightarrow$ proof goal