Did I get it right?

COS 326
David Walker
Princeton University

http://.../cos326/notes/evaluation.php
http://.../cos326/notes/reasoning.php
“Did I get it right?”

– Most fundamental question you can ask about a computer program

**Techniques for answering:**

**Grading**
- hand in program to TA
- check to see if you got an A
- (does not apply after school is out)

**Testing**
- create a set of sample inputs
- run the program on each input
- check the results
- how far does this get you?
  - has anyone ever tested a homework and not received an A?
  - why did that happen?

**Proving**
- consider all legal inputs
- show every input yields correct result
- how far does this get you?
  - has anyone ever proven a homework correct and not received an A?
  - why did that happen?
Program proving

• The basic, overall *mechanics* of proving functional programs correct is not particularly hard.
  – You are already doing it to some degree.
  – The real goal of this lecture to help you further organize your thoughts and to give you a more systematic means of understanding your programs.
  – Of course, it can certainly be hard to prove some specific program has some specific property -- just like it can be hard to write a program that solves some hard problem

• We are going to focus on proving the correctness of *pure expressions*
  – their meaning is determined exclusively by the value they return
  – don’t print, don’t mutate global variables, don’t raise exceptions
  – always terminate
  – another word for “pure expression” is “valuable expression”
“Expressions always terminate”

Two key concepts:

– A **valuable expression**
  * an expression that always terminates and produces a value

– A **total function** with type t1 -> t2
  * a function that terminates on all arguments with type t1, producing a value of type t2
  * the “opposite” of a total function is a **partial function**
    – terminates on some (possibly all) input values

Many reasoning rules depend on expressions being valuable and hence the functions that are applied being total.

*Unless told otherwise*, you can assume functions are total and expressions are valuable. (Such facts can typically be proven by induction.)
Example Theorems

We'll prove properties of OCaml expressions, starting with equivalence properties:

**Theorem**: easy 1 20 30 == 50

**Theorem**: for all natural numbers $n$, $\exp n == 2^n$

**Theorem**: for all lists $xs, ys$, $\text{length} (\text{cat} xs ys) == \text{length} xs + \text{length} ys$

```ocaml
let easy x y z = x * (y + z)

let rec exp n = match n with
  | 0 -> 1
  | n -> 2 * exp (n-1)

let rec length xs = match xs with
  | [] => 0
  | x::xs => 1 + length xs

let rec cat xs1 xs2 = match xs with
  | [] -> xs2
  | hd::tl -> hd :: cat tl xs2
```