Recitation 2

Wednesday 24 January 2018
This recitation talks about pattern matching, functions as values, and inductive proof.

Pattern Matching

What stuff can you pattern match on?

- constants (1,2,3...)
- variables
- constructors
- tuples
- wildcards

What stuff you cannot pattern match on?

- non-equality types (e.g. real)
- function applications (e.g. x + y)

> Examples

```
fun foo (0:int) : int = 1     (* matching a constant *)
  | foo (x:int) : int = x - 1 (* matching a variable *)

fun bar ([]:int list) : int = 0             (* matching a constant *)
  | bar (x::L : int list) : int = x + bar L (* matching a constructor *)

(* returns (x <= y) *)
fun cmp (0:int, _:int) : bool = true (* matching constant and wildcard *)
  | cmp (_, 0) = false               (* same *)
  | cmp (x, y) = cmp (x - 1, y - 1)  (* matching variables *)
```

Functions are Values

```
fn x:int => x + (2 + 3) (* function 1 *)
fn x:int => x + 5       (* function 2 *)
```

> The above two functions are different values. They do not "reduce" to one another. They are "extensionally equivalent." This means for all values to which the functions are applied, the application reduces to the same values.

Induction Proof

```
fun fact(0:int):int = 1
  | fact(x) = x * fact(x - 1)

fun fact'(0:int, acc:int):int = acc
  | fact'(x, acc) = fact'(n - 1, n * acc)
```

Proof: `acc * fact(x) === fact'(x, acc)` .

Proof by induction on `x` .

B.C.: `x = 0` .

`acc * fact(0) === acc * 1 = acc` (clause 1 of `fact`)
`fact'(0, acc) === acc` (clause 1 of `fact'`)

I.H.: `acc * fact(x) === fact'(x, acc)` for some `x` and for all `acc` .

W.T.S.: `acc * fact(x + 1) === fact'(x + 1, acc)` for all `acc` .

```
acc * fact(x + 1) === acc * (x + 1) * fact(x) (clause 2 of fact)
                  === (x + 1) * acc * fact(x)
                  === fact'(x, (x + 1) * acc) (I.H.)
                  === fact'(x + 1, acc)       (clause 2 of fact')
```