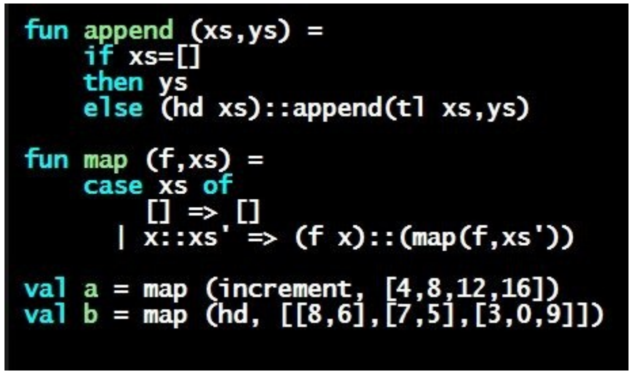

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

fun map (f,xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f,xs'))

val a = map (increment, [4,8,12,16])
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman
2013

Mutual Recursion

Mutual Recursion

- Allow **`f`** to call **`g`** and **`g`** to call **`f`**
- Useful? Yes.
  - Idiom we will show: implementing state machines
- The problem: ML's bindings-in-order rule for environments
  - Fix #1: Special new language construct
  - Fix #2: Workaround using higher-order functions

New language features

- Mutually recursive functions (the `and` keyword)

```
fun f1 p1 = e1
and f2 p2 = e2
and f3 p3 = e3
```

- Similarly, mutually recursive datatype bindings

```
datatype t1 = …
and t2 = …
and t3 = …
```

- Everything in "mutual recursion bundle" type-checked together and can refer to each other

State-machine example

- Each "state of the computation" is a function
  - "State transition" is "call another function" with "rest of input"
  - Generalizes to any finite-state-machine example

```
fun state1 input_left = …

and state2 input_left = …

and …
```

Work-around

- Suppose we did not have support for mutually recursive functions
  - Or could not put functions next to each other

- Can have the “later” function pass itself to the “earlier” one
  - Yet another higher-order function idiom

```
fun earlier (f,x) = … f y …

… (* no need to be nearby *)

fun later x = … earlier(later,y) …
```