```
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

ML Type Inference

Overview

- Will describe ML type inference via several examples
  - General algorithm is a slightly more advanced topic
  - Supporting nested functions also a bit more advanced

- Enough to help you “do type inference in your head”
  - And appreciate it is not magic

Key steps

- Determine types of bindings in order
  - (Except for mutual recursion)
  - So you cannot use later bindings: will not type-check

- For each **`val`** or **`fun`** binding:
  - Analyze definition for all necessary facts (constraints)
  - Example: If see **`x > 0`**, then **`x`** must have type **`int`**
  - Type error if no way for all facts to hold (over-constrained)

- Afterward, use type variables (e.g., **`'a`**) for any unconstrained types
  - Example: An unused argument can have any type

- (Finally, enforce the *value restriction*, discussed later)

Very simple example

Next segments will go much more step-by-step

- Like the automated algorithm does

```
val x = 42 (* val x : int *)

fun f (y, z, w) =
    if y (* y must be bool *)
    then z + x (* z must be int *)
    else 0 (* both branches have same type *)
(* f must return an int
   f must take a bool * int * ANYTHING
   so val f : bool * int * 'a -> int
 *)
```

Relation to Polymorphism

- Central feature of ML type inference: it can infer types with type variables
  - Great for code reuse and understanding functions

- But remember there are two orthogonal concepts
  - Languages can have type inference without type variables
  - Languages can have type variables without type inference