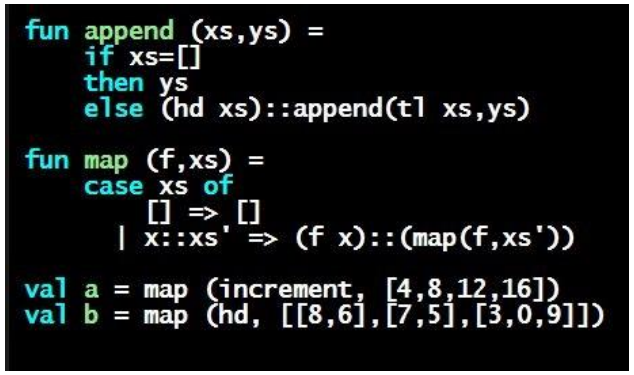

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Soundness and Completeness

Correctness

Suppose a type system is supposed to prevent X for some X

- A type system is *sound* if it never accepts a program that, when run with some input, does X
  - No *false negatives*

- A type system is *complete* if it never rejects a program that, no matter what input it is run with, will not do X
  - No *false positives*

The goal is usually for a PL type system to be sound (so you can rely on it) but not complete

  - "Fancy features" like generics aimed at "fewer false positives"

Notice soundness/completeness is with respect to X

Incompleteness

A few functions ML rejects even though they do not divide by a string

```
fun f1 x = 4 div "hi"  (* but f1 never called *)

fun f2 x = if true then 0 else 4 div "hi"

fun f3 x = if x then 0 else 4 div "hi"
val x = f3 true

fun f4 x = if x <= abs x then 0 else 4 div "hi"

fun f5 x = 4 div x
val y = f5 (if true then 1 else "hi")
```

Why incompleteness

- Almost anything you might like to check statically is undecidable:
  - Any static checker *cannot* do all of: (1) always terminate, (2) be sound, (3) be complete
  - This is a mathematical theorem!
- Examples:
  - Will this function terminate on some input?
  - Will this function ever use a variable not in the environment?
  - Will this function treat a string as a function?
  - Will this function divide by zero?
- Undecidability is an essential concept at the core of computing
  - The inherent approximation of static checking is probably its most important ramification

What about unsoundness?

Suppose a type system were unsound. What could the PL do?

- Fix it with an updated language definition?
- Insert dynamic checks as needed to prevent X from happening?
- Just allow X to happen even if "tried to stop it"?
- Worse: Allow not just X, but *anything* to happen if "programmer gets something wrong"
  - Will discuss C and C++ next…