

```
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

Optional: Are Closures Efficient?

Is that expensive?

- *Time* to build a closure is tiny: a struct with two fields
- *Space* to store closures *might* be large if environment is large
 - But environments are immutable, so natural and correct to have lots of sharing, e.g., of list tails (cf. lecture 3)
 - Still, end up keeping around bindings that are not needed
- Alternative used in practice: When creating a closure, store a possibly-smaller environment holding only the variables that are **free variables** in the function body
 - Free variables: Variables that occur, not counting shadowed uses of the same variable name
 - A function body would never need anything else from the environment

Free variables examples

```
(lambda () (+ x y z)) ; {x, y, z}
```

```
(lambda (x) (+ x y z)) ; {y, z}
```

```
(lambda (x) (if x y z)) ; {y, z}
```

```
(lambda (x) (let ([y 0]) (+ x y z))) ; {z}
```

```
(lambda (x y z) (+ x y z)) ; {}
```

```
(lambda (x) (+ y (let ([y z]) (+ y y)))) ; {y, z}
```

Computing free variables

- So does the interpreter have to analyze the code body every time it creates a closure?
- No: Before evaluation begins, compute free variables of every function in program and store this information with the function
- Compared to naïve store-entire-environment approach, building a closure now takes more time but less space
 - And time proportional to number of free variables
 - And various optimizations are possible
- [Also use a much better data structure for looking up variables than a list]

Compiling higher-order functions

[This is extra-optional]

- If we are compiling to a language without closures (like assembly), cannot rely on there being a “current environment”
- So compile functions by having the translation produce “regular” functions that *all* take an *extra explicit argument* called “environment”
- And compiler replaces all uses of free variables with code that looks up the variable using the environment argument
 - Can make these fast operations with some tricks
- Running program still creates closures and every function call passes the closure’s environment to the closure’s code