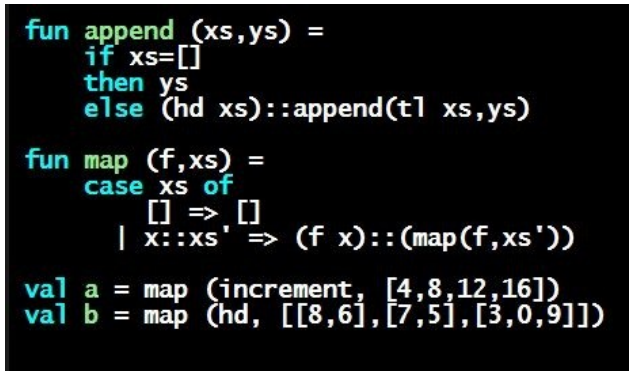

Programming Languages

Dan Grossman
2013

Lexical Scope and Higher-Order Functions

The rule stays the same

A function body is evaluated in the environment where the function was defined (created)

- Extended with the function argument

Nothing changes to this rule when we take and return functions

- But “the environment” may involve nested let-expressions, not just the top-level sequence of bindings

Makes first-class functions much more powerful

- Even if may seem counterintuitive at first

Example: Returning a function

```
(* 1 *) val x = 1
(* 2 *) fun f y =
(* 2a *)     let val x = y+1
(* 2b *)     in fn z => x+y+z end
(* 3 *) val x = 3
(* 4 *) val g = f 4
(* 5 *) val y = 5
(* 6 *) val z = g 6
```

- Trust the rule: Evaluating line 4 binds to `g` to a closure:
  - Code: "take `z` and have body `x+y+z`"
  - Environment: "`y` maps to `4`, `x` maps to `5` (shadowing), …"
  - So this closure will always add `9` to its argument
- So line 6 binds `15` to `z`

Example: Passing a function

```
(* 1 *) fun f g = (* call arg with 2 *)
(* 1a *)     let val x = 3
(* 1b *)     in g 2 end
(* 2 *) val x = 4
(* 3 *) fun h y = x + y
(* 4 *) val z = f h
```

- Trust the rule: Evaluating line 3 binds **`h`** to a closure:
  - Code: "take **`y`** and have body **`x+y`**"
  - Environment: "**`x`** maps to **`4`**, **`f`** maps to a closure, …"
  - So this closure will always add **`4`** to its argument
- So line 4 binds **`6`** to **`z`**
  - Line 1a is as stupid and irrelevant as it should be