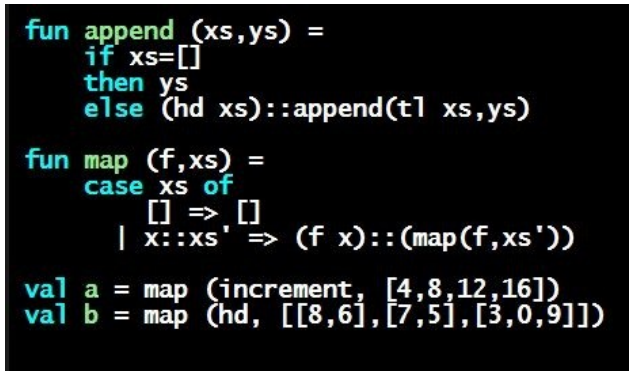

Programming Languages

Dan Grossman
2013

Another Closure Idiom: Combining Functions

More idioms

- We know the rule for lexical scope and function closures
  - Now what is it good for

A partial but wide-ranging list:

- Pass functions with private data to iterators: Done
- Combine functions (e.g., composition)
- Currying (multi-arg functions and partial application)
- Callbacks (e.g., in reactive programming)
- Implementing an ADT with a record of functions

Combine functions

Canonical example is function composition:

```
fun compose (f,g) = fn x => f (g x)
```

- Creates a closure that "remembers" what `f` and `g` are bound to
- Type `('b -> 'c) * ('a -> 'b) -> ('a -> 'c)`
  but the REPL prints something *equivalent*

- ML standard library provides this as infix operator `o`
- Example (third version best):

```
fun sqrt_of_abs i = Math.sqrt(Real.fromInt(abs i))
fun sqrt_of_abs i = (Math.sqrt o Real.fromInt o abs) i
val sqrt_of_abs = Math.sqrt o Real.fromInt o abs
```

Left-to-right or right-to-left

```
val sqrt_of_abs = Math.sqrt o Real.fromInt o abs
```

As in math, function composition is "right to left"

- "take absolute value, convert to real, and take square root"
- "square root of the conversion to real of absolute value"

"Pipelines" of functions are common in functional programming and many programmers prefer left-to-right

- Can define our own infix operator
- This one is very popular (and predefined) in F#

```
infix |>
fun x |> f = f x

fun sqrt_of_abs i =
   i |> abs |> Real.fromInt |> Math.sqrt
```

Another example

- "Backup function"

```
fun backup1 (f,g) =
    fn x => case f x of
              NONE => g x
            | SOME y => y
```

- As is often the case with higher-order functions, the types hint at what the function does

```
('a -> 'b option) * ('a -> 'b) -> 'a -> 'b
```