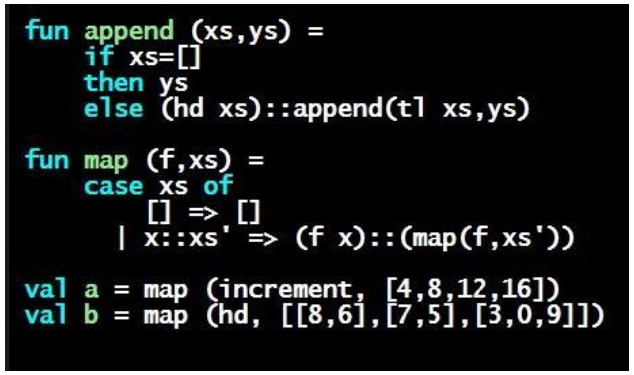

Programming Languages

Dan Grossman
2013

Partial Application

"Too Few Arguments"

- Previously used currying to simulate multiple arguments
- But if caller provides "too few" arguments, we get back a closure "waiting for the remaining arguments"
  - Called partial application
  - Convenient and useful
  - Can be done with any curried function
- No new semantics here: a pleasant idiom

Example

```
fun fold f acc xs =
    case xs of
      []     => acc
    | x::xs' => fold f (f(acc,x)) xs'

fun sum_inferior xs = fold (fn (x,y) => x+y) 0 xs

val sum = fold (fn (x,y) => x+y) 0
```

As we already know, **`fold (fn (x,y) => x+y) 0`**
evaluates to a closure that given **`xs`**, evaluates the case-expression with **`f`** bound to **`fold (fn (x,y) => x+y)`** and **`acc`** bound to **`0`**

Unnecessary function wrapping

```
fun sum_inferior xs = fold (fn (x,y) => x+y) 0 xs

val sum = fold (fn (x,y) => x+y) 0
```

- Previously learned not to write `fun f x = g x` when we can write `val f = g`

- This is the same thing, with `fold (fn (x,y) => x+y) 0` in place of `g`

Iterators

- Partial application is particularly nice for iterator-like functions
- Example:

```
fun exists predicate xs =
    case xs of
      []     => false
    | x::xs' => predicate x
                orelse exists predicate xs'

val no = exists (fn x => x=7) [4,11,23]
val hasZero = exists (fn x => x=0)
```

- For this reason, ML library functions of this form usually curried
  - Examples: **`List.map`**, **`List.filter`**, **`List.foldl`**

The Value Restriction Appears ☹

If you use partial application to *create a polymorphic function*, it may not work due to the value restriction

- Warning about "type vars not generalized"
  - And won't let you call the function
- This should surprise you; you did nothing wrong ☺ but you still must change your code
- See the code for workarounds
- Can discuss a bit more when discussing type inference