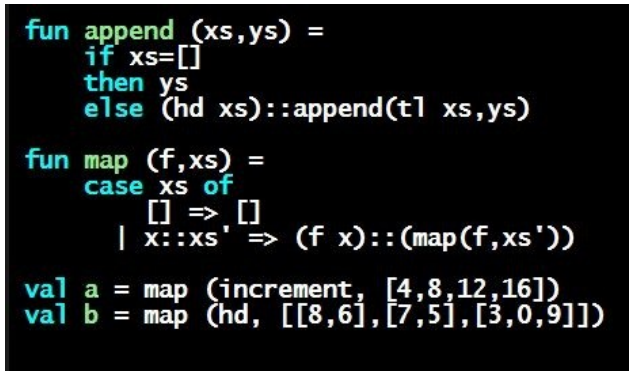

Programming Languages

Dan Grossman
2013

Lists and Options are Datatypes

Recursive datatypes

Datatype bindings can describe recursive structures

- Have seen arithmetic expressions
- Now, linked lists:

```
datatype my_int_list = Empty
                     | Cons of int * my_int_list

val x = Cons(4,Cons(23,Cons(2008,Empty)))

fun append_my_list (xs,ys) =
    case xs of
        Empty => ys
      | Cons(x,xs') => Cons(x,
   append_my_list(xs',ys))
```

Options are datatypes

Options are just a predefined datatype binding

- **`NONE`** and **`SOME`** are *constructors*, not just functions
- So use pattern-matching not **`isSome`** and **`valOf`**

```
fun inc_or_zero intoption =
    case intoption of
         NONE => 0
          | SOME i => i+1
```

Lists are datatypes

Do not use **`hd`**, **`tl`**, or **`null`** either

- `[]` and `::` are constructors too
- (strange syntax, particularly *infix*)

```
fun sum_list xs =
    case xs of
         [] => 0
          | x::xs' => x + sum_list xs'

fun append (xs,ys) =
    case xs of
        [] => ys
          | x::xs' => x :: append(xs',ys)
```

Why pattern-matching

- Pattern-matching is better for options and lists for the same reasons as for all datatypes
  - No missing cases, no exceptions for wrong variant, etc.

- We just learned the other way first for pedagogy
  - Do not use **`isSome`**, **`valOf`**, **`null`**, **`hd`**, **`tl`** on Homework 2

- So why are **`null`**, **`tl`**, etc. predefined?
  - For passing as arguments to other functions (next week)
  - Because sometimes they are convenient
  - But not a big deal: could define them yourself