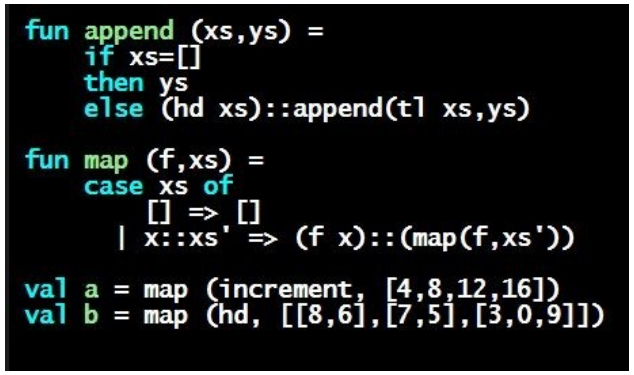

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

2013

Polymorphic Types and Equality Types

An example

- "Write a function that appends two string lists"

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

- You expect `string list * string list -> string list`
- Implementation says `'a list * 'a list -> 'a list`
- This is okay [such as on your homework]: why?

More general

The type

```
'a list * 'a list -> 'a list
```

is more general than the type

```
string list * string list -> string list
```

- It "can be used" as any less general type, such as
  ```
  int list * int list -> int list
  ```

- But it is not more general than the type
  ```
  int list * string list -> int list
  ```

The "more general" rule

Easy rule you (and the type-checker) can apply without thinking:

A type *t1* is more general than the type *t2* if you can take *t1*, replace its type variables consistently, and get *t2*

- Example: Replace each `'a` with `int * int`
- Example: Replace each `'a` with `bool` and each `'b` with `bool`
- Example: Replace each `'a` with `bool` and each `'b` with `int`
- Example: Replace each `'b` with `'a` and each `'a` with `'a`

Other rules

- Can combine the "more general" rule with rules for equivalence
  - Use of type synonyms does not matter
  - Order of field names does not matter

Example, given

```
type foo = int * int
```

the type

```
{quux : 'b, bar : int * 'a, baz : 'b}
```

is more general than

```
{quux : string, bar : foo, baz : string}
```

which is equivalent to

```
{bar : int*int, baz : string, quux : string}
```

Equality types

- You might also see type variables with a second "quote"
  - Example: **`''a list * ''a -> bool`**

- These are "equality types" that arise from using the **=** operator
  - The **=** operator works on lots of types: **`int`**, **`string`**, tuples containing all equality types, …
  - But not all types: function types, **`real`**, …

- The rules for more general are exactly the same except you have to replace an equality-type variable with a type that can be used with **=**
  - A "strange" feature of ML because **=** is special

Example

```
(* ''a * ''a -> string *)
fun same_thing(x, y) =
    if x=y then "yes" else "no"

(* int -> string *)
fun is_three x =
    if x=3 then "yes" else "no"
```

(You can ignore the warning about “calling polyEqual”)