

# Recitation #3

E Section • • 1/31/2006

# *Lists*

- datatype 'a list = nil | :: of 'a \* 'a list
- 1::2::3::4
- fun doubleup(nil) = nil  
| doubleup(x::l) = (x,x)::doubleup(l)

# Lists


– datatype 'a list = nil | :: of 'a \* 'a list

– 1::2::3::4


– fun doubleup(nil) = nil  
| doubleup(x::l) = (x,x)::doubleup(l)

# Lists

– datatype 'a list = nil | :: of 'a \* 'a list




– 1::2::3::4




– fun doubleup(nil) = nil  
| doubleup(x::l) = (x,x)::doubleup(l)

# Lists


– datatype 'a list = nil | :: of 'a \* 'a list



– 1::2::3::4



– fun doubleup(nil) = nil  
| doubleup(x::l) = (x,x)::doubleup(l)



# List References

- Hansen & Rischel (course textbook) 5.6
- <http://www.standardml.org/Basis/>

# List exercise 1

```
(* val hd : 'a list -> 'a
   hd(l) returns the first element
       in list l.
   Invariant: l is nonempty.
   Effects: none.
*)
```

# List exercise 1

```
(* val hd : 'a list -> 'a
   hd(l) returns the first element
       in list l.
   Invariant: l is nonempty.
   Effects: none.
*)
fun hd (x::_) = x
```

# List exercise 1

```
(* val hd : 'a list -> 'a
   hd(l) returns the first element
       in list l.
   Invariant: l is nonempty.
   Effects: none.
*)
fun hd (x::_) = x
```

```
stdin:27.5-27.17 Warning: match nonexhaustive
      x :: _ => ...
```

# List exercise 2

```
(* val null : `a list -> bool
   null(l) returns true iff l is
           the empty list.
   Invariants: none.
   Effects: none.
*)
```

# List exercise 2

```
(* val null : `a list -> bool
   null(l) returns true iff l is
           the empty list.
   Invariants: none.
   Effects: none.
*)
fun null nil      = true
  | null (_::_) = false
```

# List exercise 3

```
(* val last : 'a list -> 'a
   last(l) returns the last
                element in list l.
   Invariant: l is nonempty.
   Effects: none.
*)
```

# List exercise 3

```
(* val last : 'a list -> 'a
   last(l) returns the last
               element in list l.
   Invariant: l is nonempty.
   Effects: none.
*)
fun last (x::nil) = x
  | last (x::l)   = last l
```

# List exercise 3

```
(* val last : 'a list -> 'a
   last(l) returns the last
           element in list l.
   Invariant: l is nonempty.
   Effects: none.
```

```
*)
```

```
fun last (x::nil) = x
  | last (x::l)   = last l
```

```
stdin:33.5-34.26 Warning: match nonexhaustive
  x :: nil => ...
  x :: l  => ...
```

# Exercises don't stop (4)

```
(* val take : 'a list * int ->
    'a list
   take(l,i) returns the list of
      the first i elements of l.
   Invariants and Effects: none.
*)
```

# Exercises don't stop (4)

```
(* val take : 'a list * int ->
                               'a list
   take(l,i) returns the list of
   the first i elements of l.
   Invariants and Effects: none.
*)
fun take (nil, i) = nil
  | take (x::l, 0) = nil
  | take (x::l, i) = x::take(l, i-1)
```

# Penultimate... (ex. 5)

```
(* val concat :  
    concat(l) concatenates the lists  
        in the list of lists l.  
    Invariants and Effects: none.  
*)
```

# Penultimate... (ex. 5)

```
(* val concat : 'a list list ->
    'a list
    concat(l) concatenates the lists
    in the list of lists l.
    Invariants and Effects: none.
*)
```

# Penultimate... (ex. 5)

```
(* val concat : `a list list ->
    `a list
    concat(l) concatenates the lists
    in the list of lists l.
    Invariants and Effects: none.
*)
fun concat nil          = nil
  | concat (lst::l)     = lst@ (concat l)
```

# Final Challenge (6)

```
(* val zip : 'a list * 'b list ->
      ('a * 'b) list
   zip(l1,l2) returns a list of
      pairs of elems of l1 and l2.
   Invariant: len(l1) == len(l2)
   Effects: none.
*)
```

# Final Challenge (6)

```
(* val zip : 'a list * 'b list ->
      ('a * 'b) list
   zip(l1,l2) returns a list of
      pairs of elems of l1 and l2.
   Invariant: len(l1) == len(l2)
   Effects: none.
*)
fun zip(x::l1,y::l2) =
      (x,y)::zip(l1,l2)
  | zip _      = nil
```

# Equality Types

# Equality types

- `int`, `char`, `string`, `bool`
- *Structured values*: checks corresponding components.
- Thus: tuples, lists, “records”, and *datatypes* built over basic types.

# NOT equality types

- Functions
- `real`
- abstract types

# Equality type variables

- `''a` instead of ``a`
- Any type that “fills in” `''a` must be an equality type.
- `op = ↪ fn : ''a * ''a -> bool`

# SML's deductive power

```
infix isin;  
fun (x isin nil)      = false  
  | (x isin (y::l)) =  
      if (x=y) then true  
        else (x isin l)
```

# SML's deductive power

```
infix isin;  
fun (x isin nil)      = false  
  | (x isin (y::l)) =  
    (x=y) orelse (x isin l)
```

# SML's deductive power

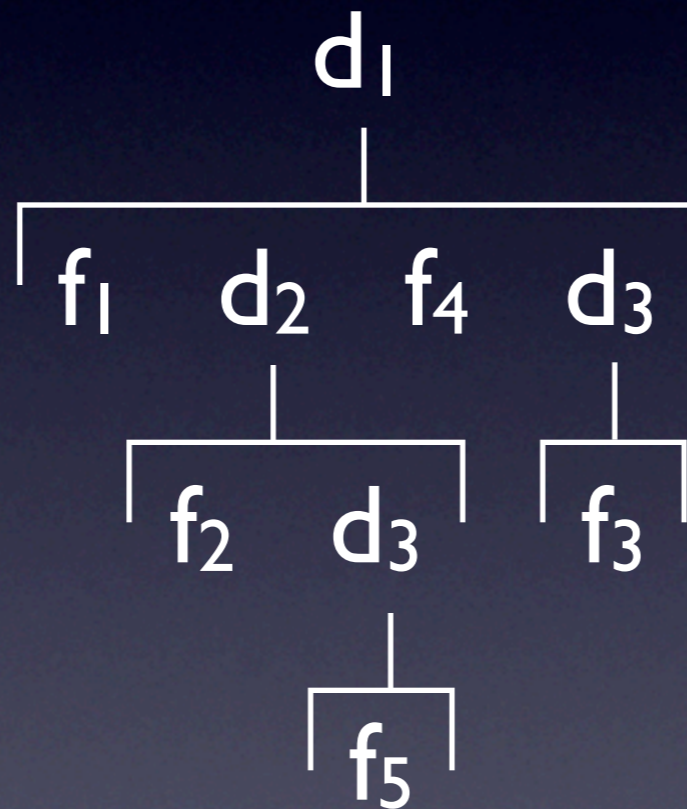
```
infix isin;  
fun (x isin nil)      = false  
  | (x isin (y::l)) =  
    (x=y) orelse (x isin l)
```

```
stdIn:68.24 Warning: calling polyEqual  
val isin = fn : 'a * 'a list -> bool
```

# Mutually recursive datatypes

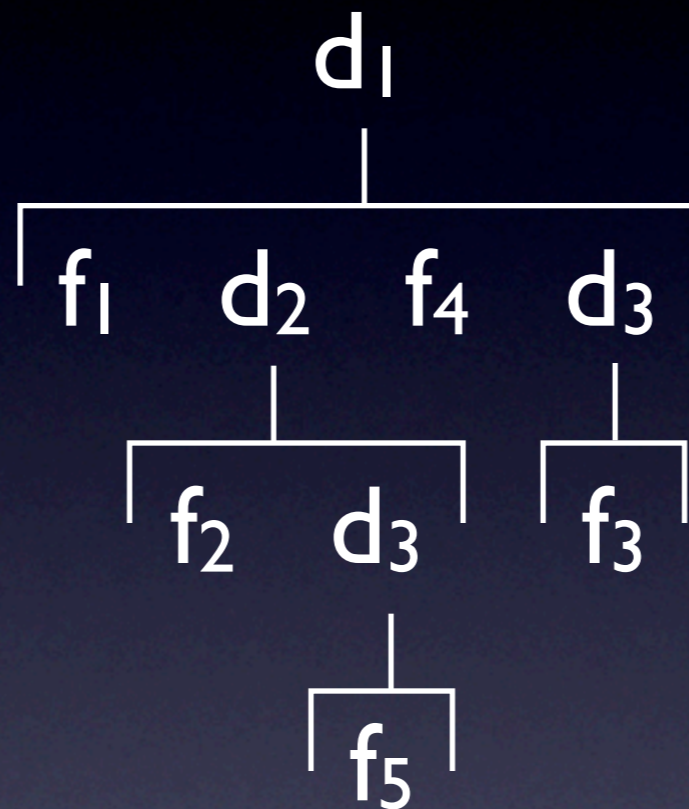
# Represent *this*:

- Filesystem:



- How?

# Represent *this*:



```
datatype elem = File of string  
           | Directory of string * contents  
withtype contents = elem list
```