Standard ML

ML Lists.1

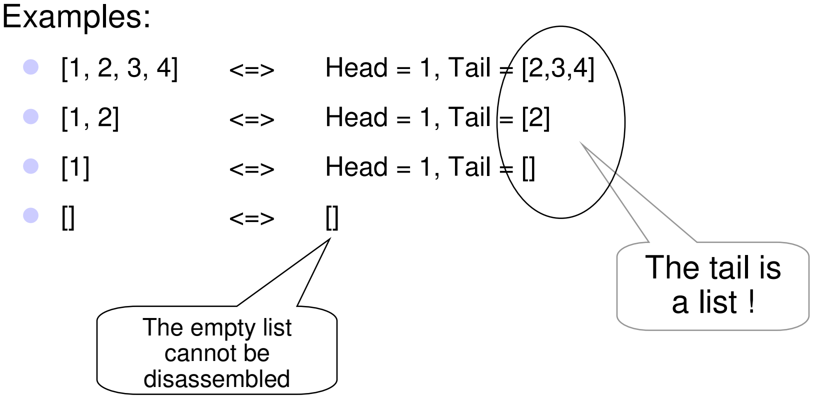
Lists

- A list is a finite sequence of elements.
  - `[3,5,9]`
  - `["a", "list" ]`
  - `[]`
- Elements may appear more than once
  - `[3,4]`
  - `[4,3]`
  - `[3,4,3]`
  - `[3,3,4]`
- Elements may have any type. But all elements of a list must have the same type.
  - `[(1,"One"),(2,"Two")] : (int*string) list`
  - `[[3.1],[],[5.7, ~0.6]]: (real list) list`
- The empty list `[]` has the polymorphic type `'a list`

ML Lists.2

Building a List

- Every list is either empty or can be constructed by joining its first element and its tail (which is a list itself)
- Examples:



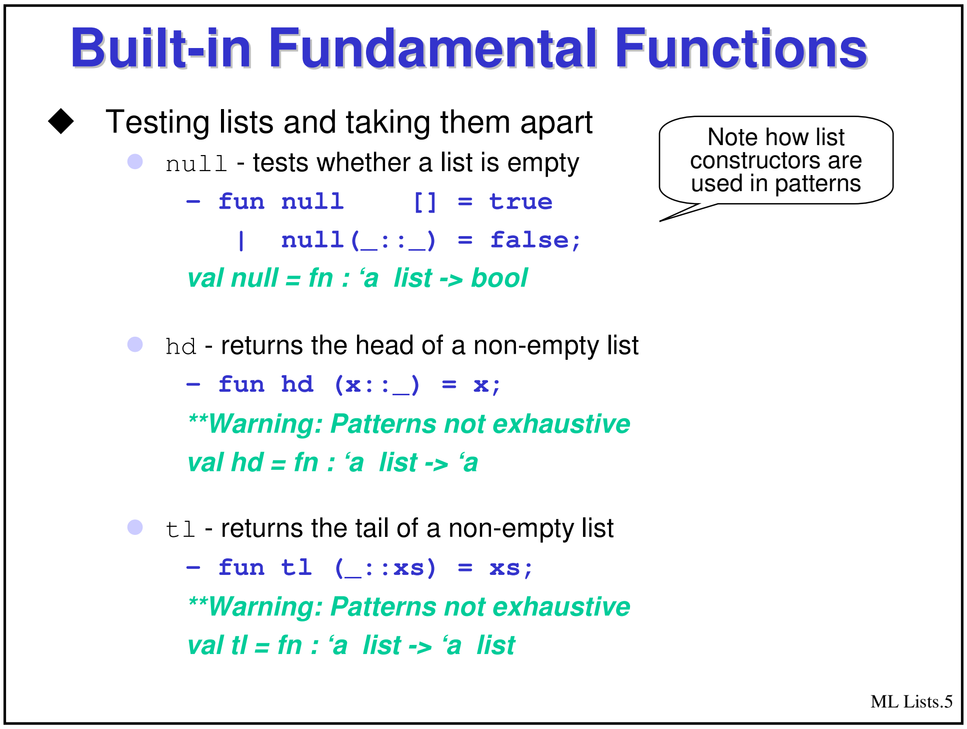
- [1, 2, 3, 4] <=> Head = 1, Tail = [2,3,4]
- [1, 2] <=> Head = 1, Tail = [2]
- [1] <=> Head = 1, Tail = []
- [] <=> []

The empty list cannot be disassembled

The tail is a list !

Building a List (cont.)

- ***`nil`*** is a synonym for the empty list **`[]`**
- The operator `::` (also called ***cons***) makes a list by putting an element in front of an existing list
- Every list is either ***`nil`*** or has the form `x::xs` where `x` is its head and `xs` is its tail (which is a list itself).
- The infix operator `::` groups to the *right*.
- The notation `[x1,x2,...,xn]` stands for `x1 :: x2 :: ... :: xn :: nil`
  - `3::(5::(9::nil))` is `[3,5,9]`
- Clarification:
  - You can build a list either with `nil` and `op::` (the list constructors) or using the brackets notation ([]) as a shortcut.



Built-in Fundamental Functions

- Testing lists and taking them apart
  - `null` - tests whether a list is empty

```
- fun null     [] = true
    |  null(_::_) = false;
```

val null = fn : 'a list -> bool

Note how list constructors are used in patterns

  - `hd` - returns the head of a non-empty list

```
- fun hd (x::_) = x;
```

***Warning: Patterns not exhaustive*

val hd = fn : 'a list -> 'a

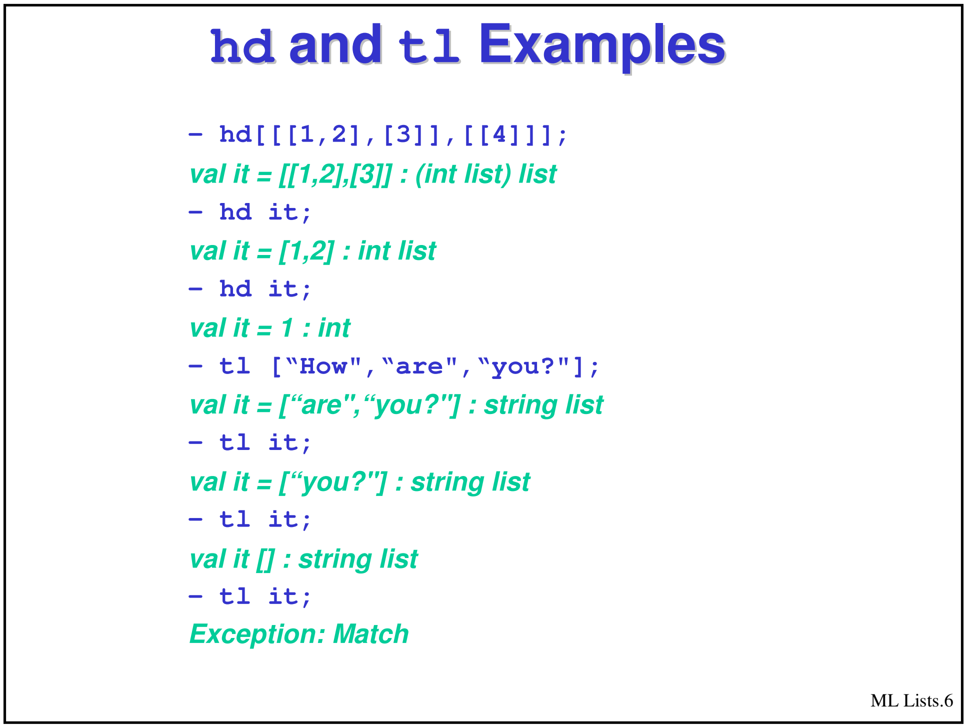
  - `tl` - returns the tail of a non-empty list

```
- fun tl (_::xs) = xs;
```

***Warning: Patterns not exhaustive*

val tl = fn : 'a list -> 'a list

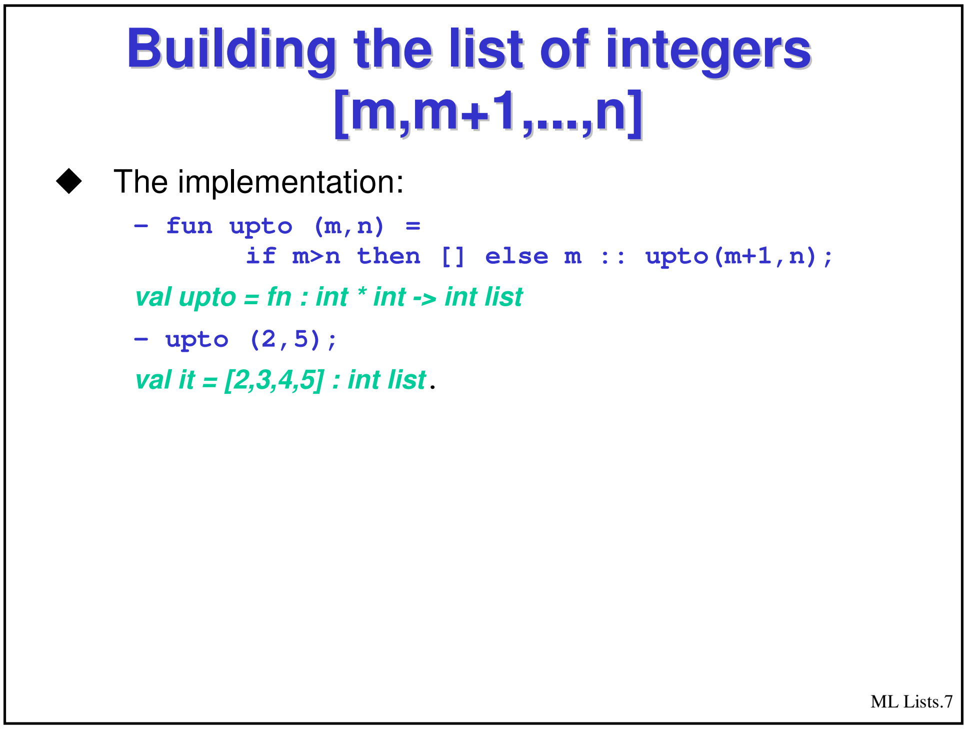
ML Lists.5



`hd` and `tl` Examples

```
- hd[[[1,2],[3]],[[4]]];
val it = [[1,2],[3]] : (int list) list
- hd it;
val it = [1,2] : int list
- hd it;
val it = 1 : int
- tl ["How","are","you?"];
val it = ["are","you?"] : string list
- tl it;
val it = ["you?"] : string list
- tl it;
val it [] : string list
- tl it;
Exception: Match
```

ML Lists.6



Building the list of integers [m,m+1,…,n]

- The implementation:

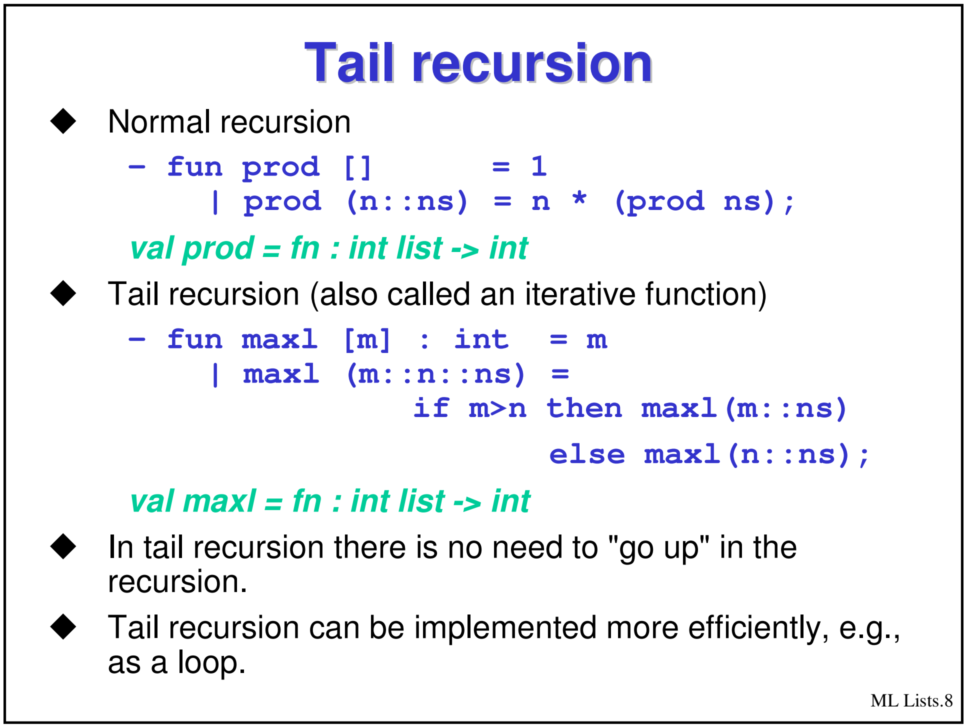
```
- fun upto (m,n) =
       if m>n then [] else m :: upto(m+1,n);
```

*val upto = fn : int * int -> int list*

```
- upto (2,5);
```

val it = [2,3,4,5] : int list.

ML Lists.7



Tail recursion

- Normal recursion

```
- fun prod []       = 1
    | prod (n::ns) = n * (prod ns);
```

val prod = fn : int list -> int

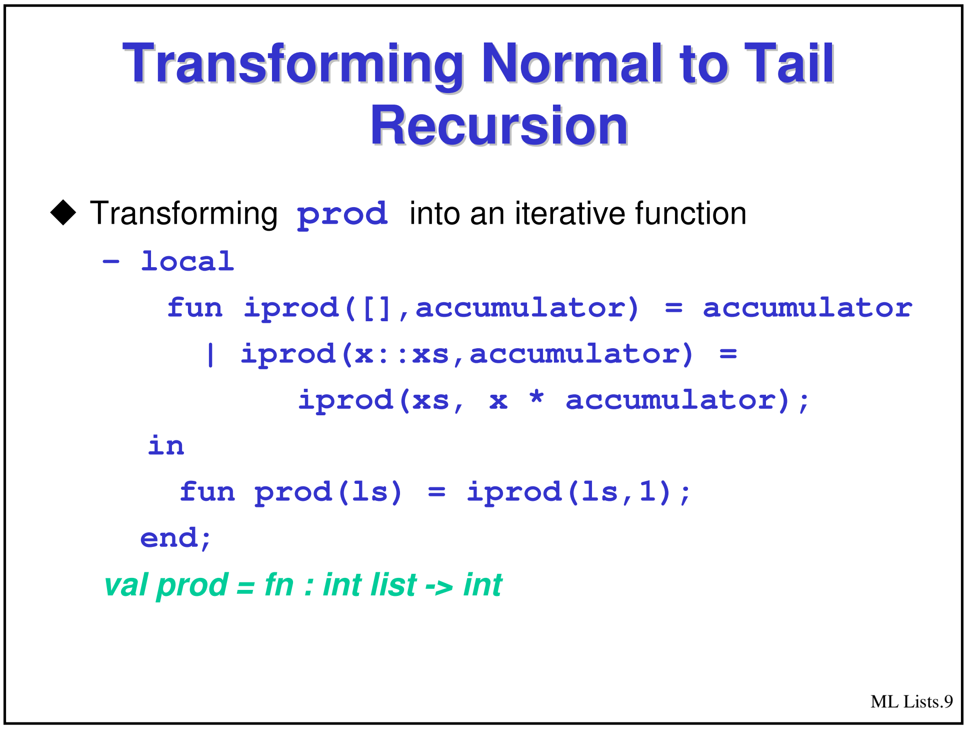
- Tail recursion (also called an iterative function)

```
- fun maxl [m] : int  = m
    | maxl (m::n::ns) =
               if m>n then maxl(m::ns)
                      else maxl(n::ns);
```

val maxl = fn : int list -> int

- In tail recursion there is no need to "go up" in the recursion.
- Tail recursion can be implemented more efficiently, e.g., as a loop.

ML Lists.8

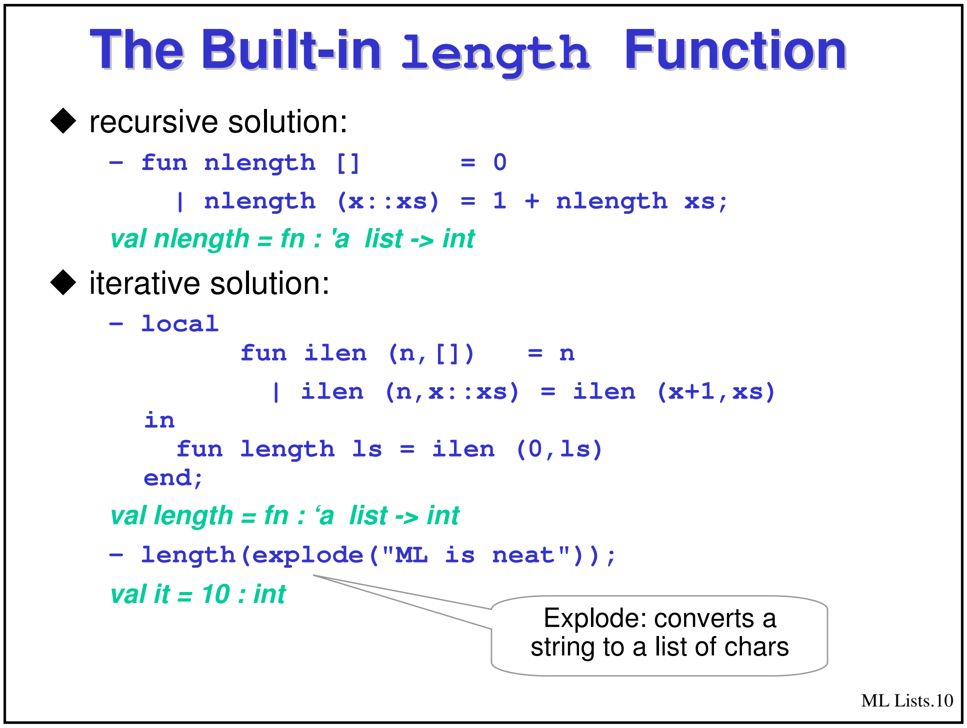


Transforming Normal to Tail Recursion

- Transforming `prod` into an iterative function

```
- local
    fun iprod([],accumulator) = accumulator
      | iprod(x::xs,accumulator) =
            iprod(xs, x * accumulator);
  in
    fun prod(ls) = iprod(ls,1);
  end;
val prod = fn : int list -> int
```

ML Lists.9



The Built-in `length` Function

- recursive solution:

```
- fun nlength []       = 0
    | nlength (x::xs) = 1 + nlength xs;
val nlength = fn : 'a  list -> int
```

- iterative solution:

```
- local
        fun ilen (n,[])    = n
          | ilen (n,x::xs) = ilen (x+1,xs)
  in
    fun length ls = ilen (0,ls)
  end;
val length = fn : ‘a  list -> int
- length(explode("ML is neat"));
val it = 10 : int
```

Explode: converts a string to a list of chars

ML Lists.10

take and drop

$$\mathrm{xs} = [\underbrace{x_1, x_2, x_3, \ldots, x_i}_{\mathrm{take(i,xs)}},\ \underbrace{x_{i+1}, \ldots, x_n}_{\mathrm{drop(i,xs)}}]$$

- **take(i,l)** returns the list of the first **i** elements of **l**

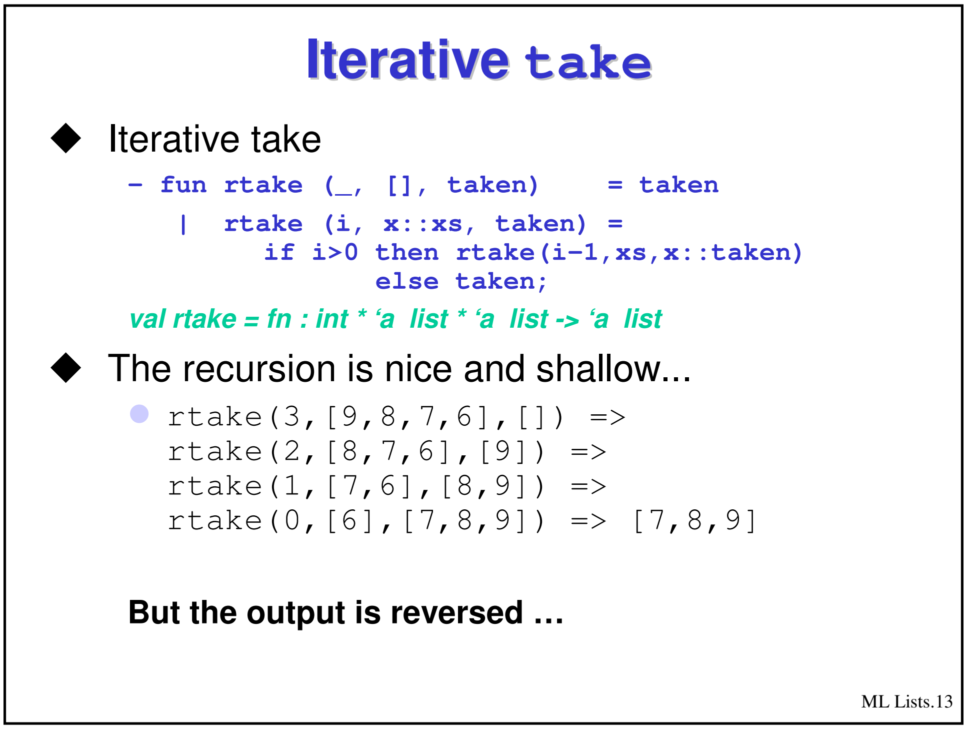
```
- fun take (i,[])     = []
    | take (i,x::xs) =
        if i>0 then x :: take(i-1,xs)
               else [];
val take = fn : int * 'a  list -> 'a  list
- take (5,explode "Throw Pascal to the dogs!");
> [#"T", #"h", #"r", #"o", #"w"] : char list
```

ML Lists.11

The Computation of take

```
take(3,[9,8,7,6]) =>
9::take(2,[8,7,6]) =>
9::(8::take(1,[7,6])) =>
9::(8::(7::take(0,[6]))) =>
9::(8::(7::[])) =>
9::(8::[7]) =>
9::[8,7] => [9,8,7]
```

ML Lists.12



Iterative take

- Iterative take

```
- fun rtake (_, [], taken)     = taken
    |  rtake (i, x::xs, taken) =
         if i>0 then rtake(i-1,xs,x::taken)
                else taken;
```

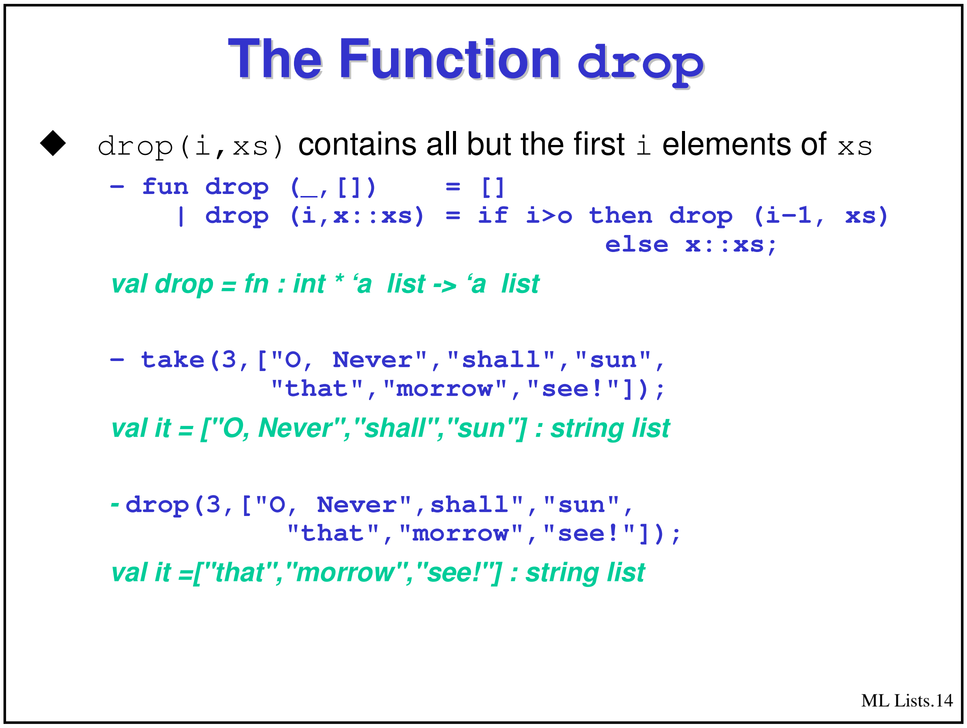
*val rtake = fn : int * ‘a list * ‘a list -> ‘a list*

- The recursion is nice and shallow...

  -
    ```
    rtake(3,[9,8,7,6],[]) =>
    rtake(2,[8,7,6],[9]) =>
    rtake(1,[7,6],[8,9]) =>
    rtake(0,[6],[7,8,9]) => [7,8,9]
    ```

But the output is reversed …

ML Lists.13



The Function drop

- `drop(i,xs)` contains all but the first `i` elements of `xs`

```
- fun drop (_,[])     = []
    | drop (i,x::xs) = if i>o then drop (i-1, xs)
                              else x::xs;
```

*val drop = fn : int * ‘a list -> ‘a list*

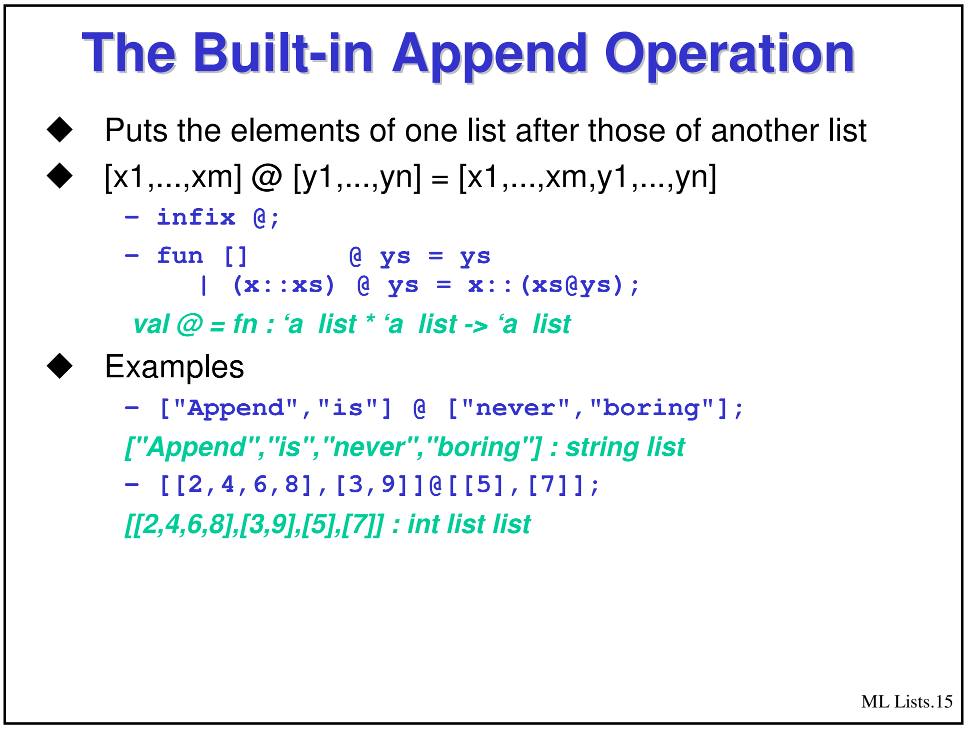
```
- take(3,["O, Never","shall","sun",
          "that","morrow","see!"]);
```

val it = ["O, Never","shall","sun"] : string list

```
-drop(3,["O, Never",shall","sun",
          "that","morrow","see!"]);
```

val it =["that","morrow","see!"] : string list

ML Lists.14



The Built-in Append Operation

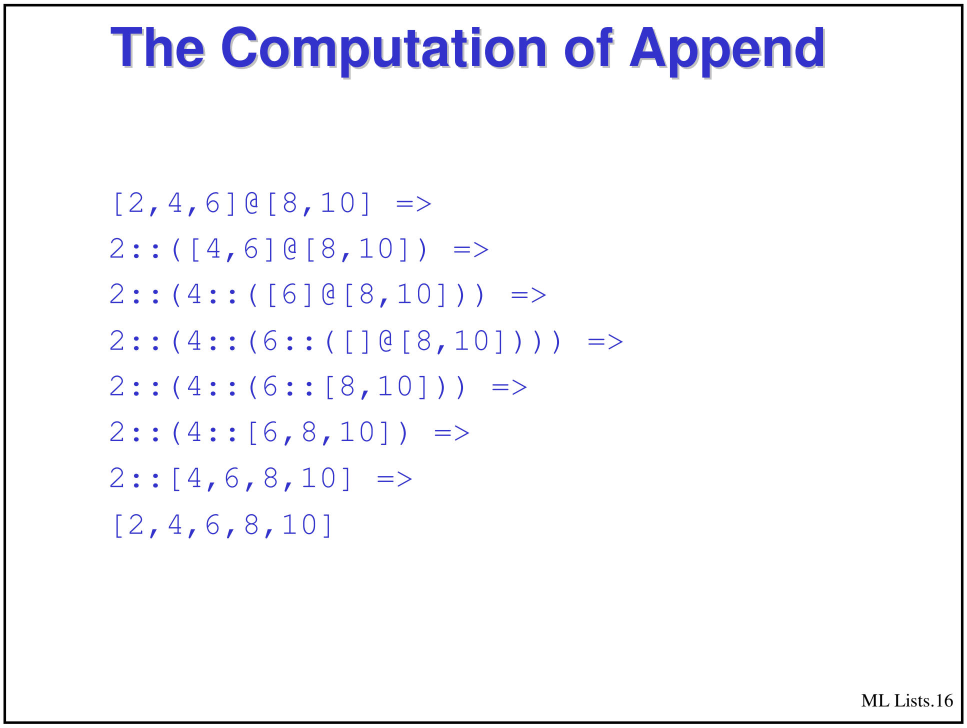
- Puts the elements of one list after those of another list
- [x1,...,xm] @ [y1,...,yn] = [x1,...,xm,y1,...,yn]

```
- infix @;
- fun []       @ ys = ys
    | (x::xs) @ ys = x::(xs@ys);
val @ = fn : ‘a  list * ‘a  list -> ‘a  list
```

- Examples

```
- ["Append","is"] @ ["never","boring"];
["Append","is","never","boring"] : string list
- [[2,4,6,8],[3,9]]@[[5],[7]];
[[2,4,6,8],[3,9],[5],[7]] : int list list
```

ML Lists.15



The Computation of Append

```
[2,4,6]@[8,10] =>
2::([4,6]@[8,10]) =>
2::(4::([6]@[8,10])) =>
2::(4::(6::([]@[8,10]))) =>
2::(4::(6::[8,10])) =>
2::(4::[6,8,10]) =>
2::[4,6,8,10] =>
[2,4,6,8,10]
```

ML Lists.16

The Built-in `rev` Function

- Using append

```
- fun nrev []       = []
    | nrev (x::xs)  = (nrev xs) @ [x];
val nrev = fn : 'a  list -> 'a  list
```

  - Append calls cons (n-1) times to copy the reversed tail of a list of length n
  - Constructing the list `[x]` calls cons again
  - Reversing the tail requires (n-1) more conses
  - The total number of conses is thus: n(n+1)/2
- Remember rtake ?...

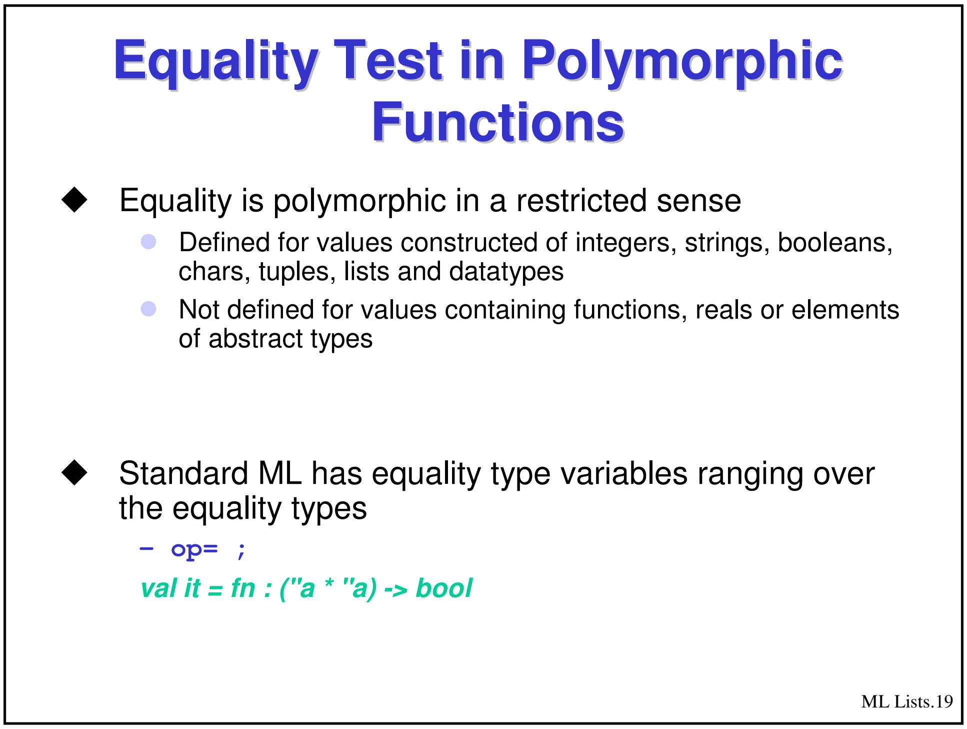
```
- local
     fun revto  ([],    ys)  = ys
       | revto  (x::xs,ys)  = revto  (xs,  x::ys);
   in fun rev xs = revto (xs,[]) end;
val rev = fn : 'a  list -> 'a  list
```

Side Note: `orelse` and `andalso`

- They are ***short-circuit*** OR and AND boolean operations.
- `B1 andalso B2` <=> `if B1 then B2    else false`
- `B1 orelse B2` <=> `if B1 then true else B2`
- Meaning the second boolean is evaluated only if needed.
- Is the following `powoftwo` function a tail recursion?

```
- fun even n =  (n mod 2 = 0);
val even = fn : int -> bool

- fun powoftwo n = (n=1) orelse
         (even(n) andalso powoftwo(n div 2));
val powoftwo = fn : int -> bool
```

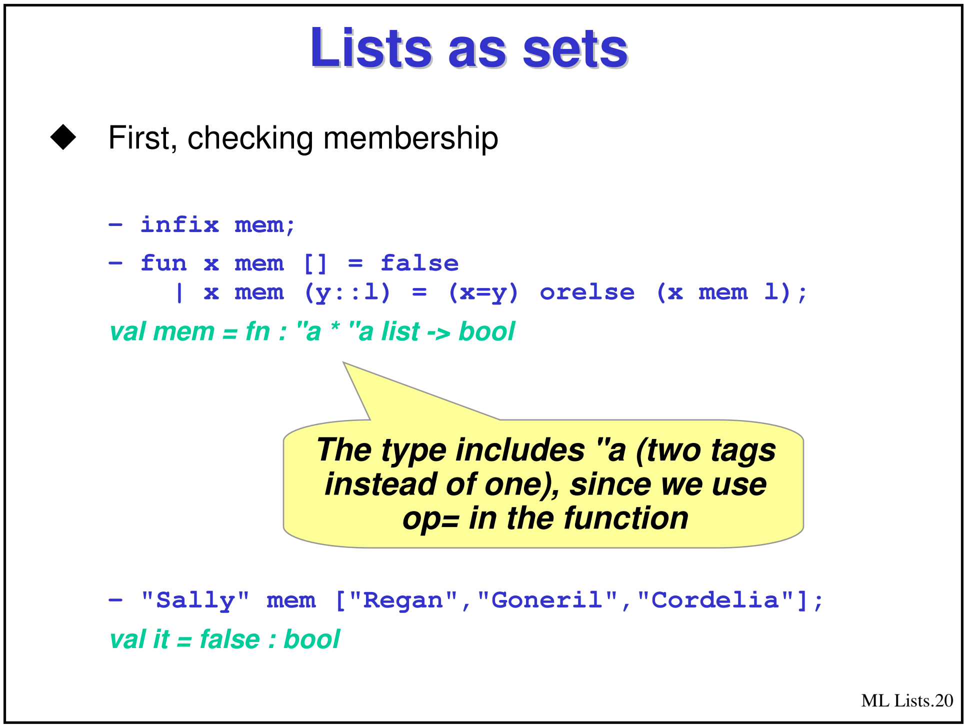


Equality Test in Polymorphic Functions

- Equality is polymorphic in a restricted sense
  - Defined for values constructed of integers, strings, booleans, chars, tuples, lists and datatypes
  - Not defined for values containing functions, reals or elements of abstract types
- Standard ML has equality type variables ranging over the equality types

```
- op= ;
val it = fn : ("a * "a) -> bool
```

ML Lists.19



Lists as sets

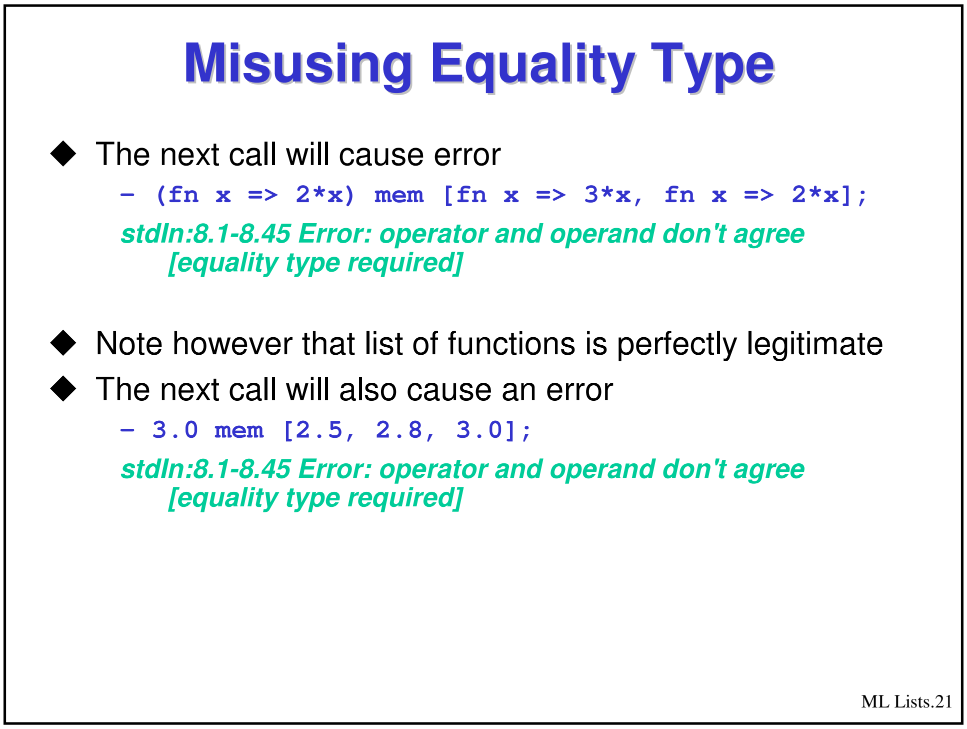
- First, checking membership

```
- infix mem;
- fun x mem [] = false
    | x mem (y::l) = (x=y) orelse (x mem l);
val mem = fn : "a * "a list -> bool
```

The type includes "a (two tags instead of one), since we use op= in the function

```
- "Sally" mem ["Regan","Goneril","Cordelia"];
val it = false : bool
```

ML Lists.20



Misusing Equality Type

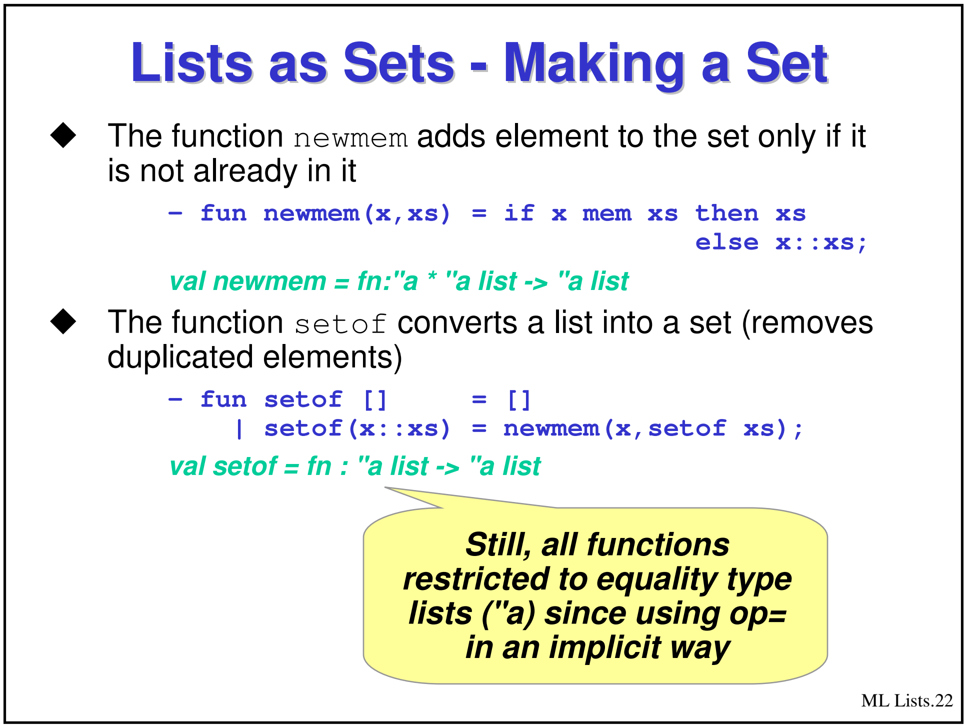
- The next call will cause error

```
- (fn x => 2*x) mem [fn x => 3*x, fn x => 2*x];
stdIn:8.1-8.45 Error: operator and operand don't agree
    [equality type required]
```

- Note however that list of functions is perfectly legitimate
- The next call will also cause an error

```
- 3.0 mem [2.5, 2.8, 3.0];
stdIn:8.1-8.45 Error: operator and operand don't agree
    [equality type required]
```

ML Lists.21



Lists as Sets - Making a Set

- The function `newmem` adds element to the set only if it is not already in it

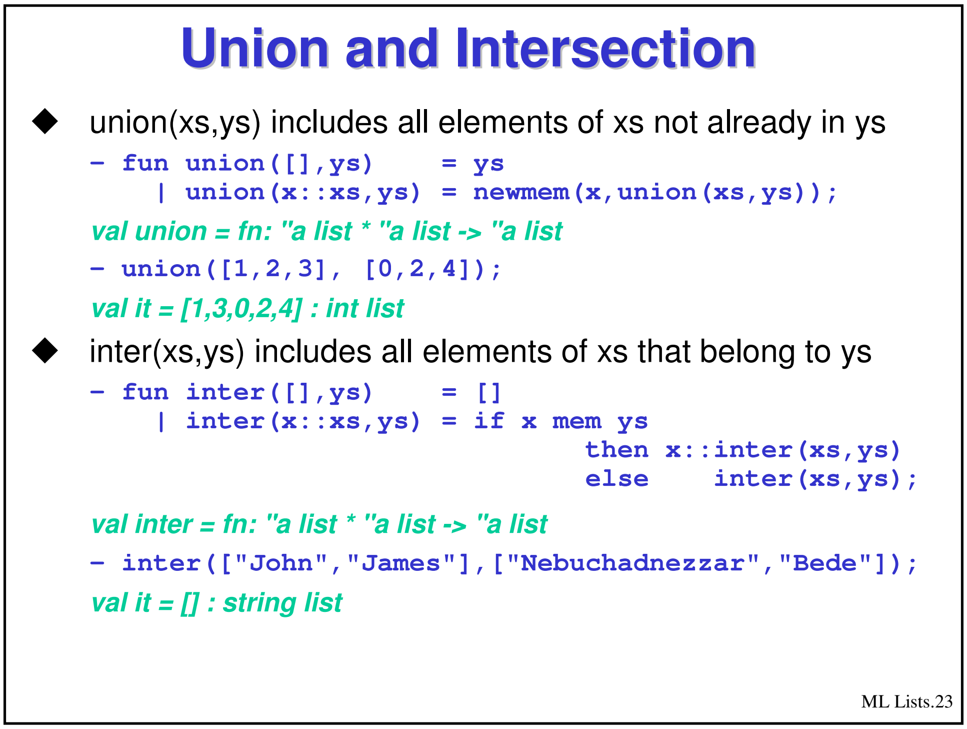
```
- fun newmem(x,xs) = if x mem xs then xs
                                 else x::xs;
val newmem = fn:''a * ''a list -> ''a list
```

- The function `setof` converts a list into a set (removes duplicated elements)

```
- fun setof []      = []
    | setof(x::xs) = newmem(x,setof xs);
val setof = fn : ''a list -> ''a list
```

Still, all functions restricted to equality type lists (''a) since using op= in an implicit way

ML Lists.22



Union and Intersection

- union(xs,ys) includes all elements of xs not already in ys

```
- fun union([],ys)     = ys
    | union(x::xs,ys) = newmem(x,union(xs,ys));
```

*val union = fn: ''a list * ''a list -> ''a list*

```
- union([1,2,3], [0,2,4]);
```

val it = [1,3,0,2,4] : int list

- inter(xs,ys) includes all elements of xs that belong to ys

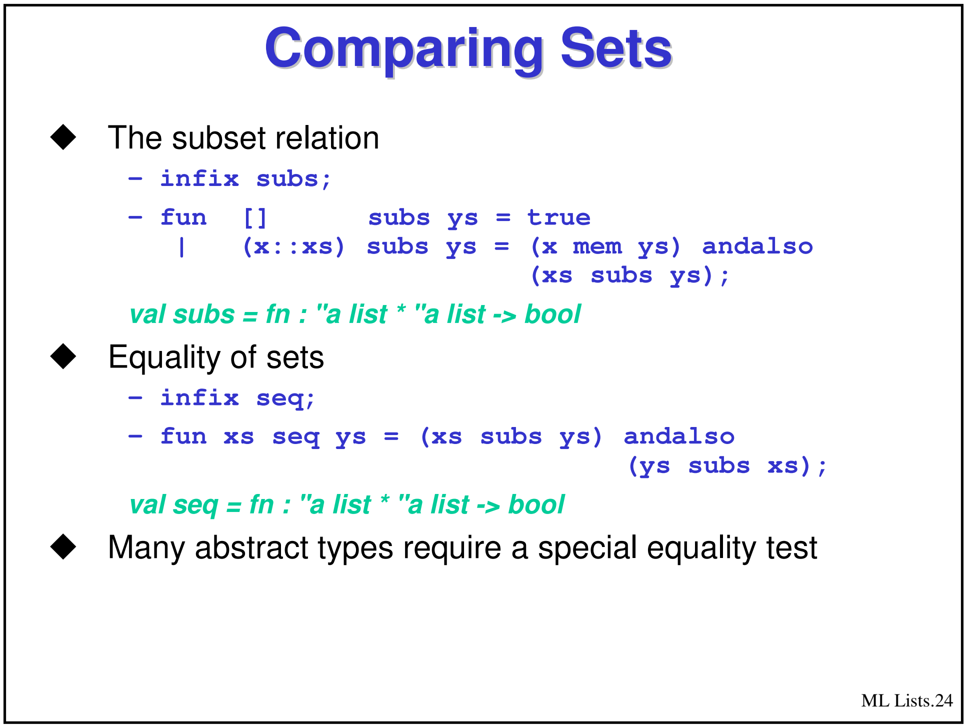
```
- fun inter([],ys)     = []
    | inter(x::xs,ys) = if x mem ys
                               then x::inter(xs,ys)
                               else    inter(xs,ys);
```

*val inter = fn: ''a list * ''a list -> ''a list*

```
- inter(["John","James"],["Nebuchadnezzar","Bede"]);
```

val it = [] : string list

ML Lists.23



Comparing Sets

- The subset relation

```
- infix subs;
- fun  []      subs ys = true
   |   (x::xs) subs ys = (x mem ys) andalso
                         (xs subs ys);
```

*val subs = fn : ''a list * ''a list -> bool*

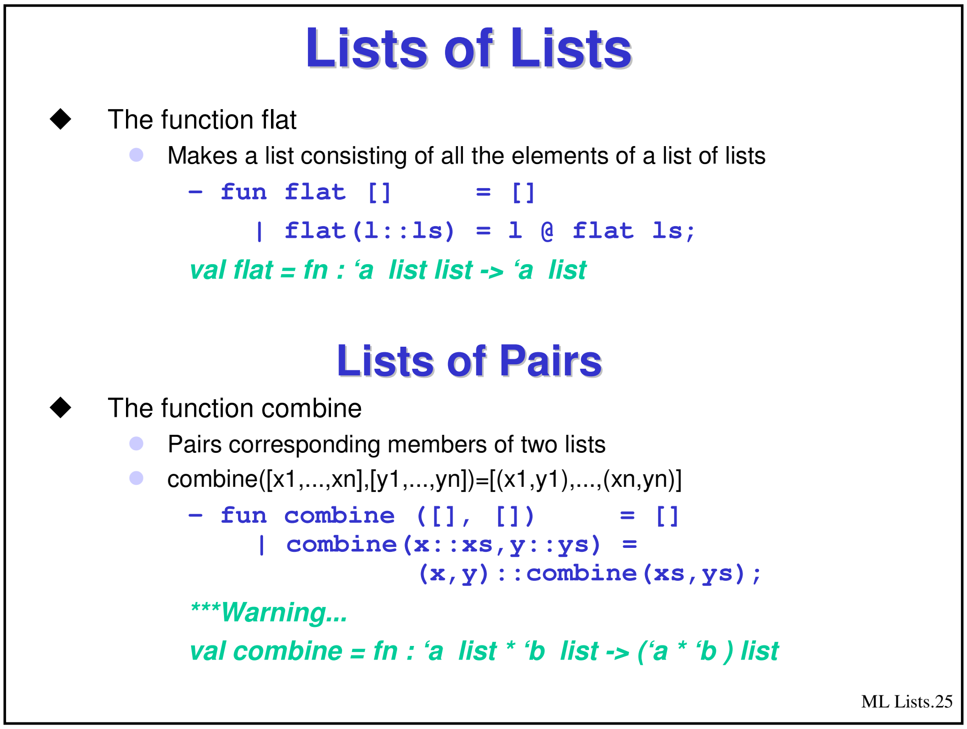
- Equality of sets

```
- infix seq;
- fun xs seq ys = (xs subs ys) andalso
                               (ys subs xs);
```

*val seq = fn : ''a list * ''a list -> bool*

- Many abstract types require a special equality test

ML Lists.24



Lists of Lists

- The function flat
  - Makes a list consisting of all the elements of a list of lists

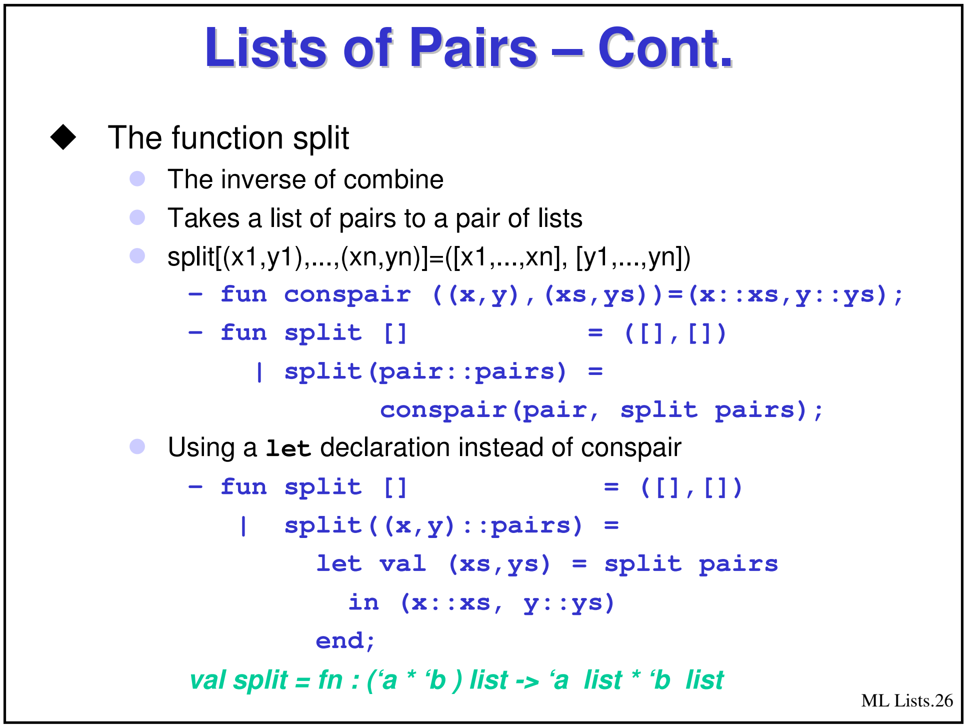
```
- fun flat []      = []
    | flat(l::ls) = l @ flat ls;
val flat = fn : 'a  list list -> 'a  list
```

Lists of Pairs

- The function combine
  - Pairs corresponding members of two lists
  - combine([x1,...,xn],[y1,...,yn])=[(x1,y1),...,(xn,yn)]

```
- fun combine ([], [])      = []
    | combine(x::xs,y::ys) =
               (x,y)::combine(xs,ys);
***Warning...
val combine = fn : 'a  list * 'b  list -> ('a * 'b ) list
```

ML Lists.25



Lists of Pairs – Cont.

- The function split
  - The inverse of combine
  - Takes a list of pairs to a pair of lists
  - split[(x1,y1),...,(xn,yn)]=([x1,...,xn], [y1,...,yn])

```
- fun conspair ((x,y),(xs,ys))=(x::xs,y::ys);
- fun split []            = ([],[])
    | split(pair::pairs) =
            conspair(pair, split pairs);
```

  - Using a `let` declaration instead of conspair

```
- fun split []              = ([],[])
   |  split((x,y)::pairs) =
         let val (xs,ys) = split pairs
           in (x::xs, y::ys)
         end;
val split = fn : ('a * 'b ) list -> 'a  list * 'b  list
```

ML Lists.26

Association Lists

- A list of (key,value) pairs
- The function `assoc` finds the value associated with a key

```
- fun assoc ([], a)            = []
    | assoc ((x,y)::pairs, a) =
        if a=x then [y] else assoc(pairs, a);
val assoc = fn:("a * 'b) list * "a -> 'b list
```

- The function `nexts` finds all successors of a node a in a directed graph represented by a list of edges (pairs)

```
- fun nexts (a,[])             = []
    | nexts (a, (x,y)::pairs) =
         if a=x then y::nexts(a,pairs)
                else    nexts(a,pairs);
val nexts = fn:"a * ("a * 'b) list -> 'b list
```

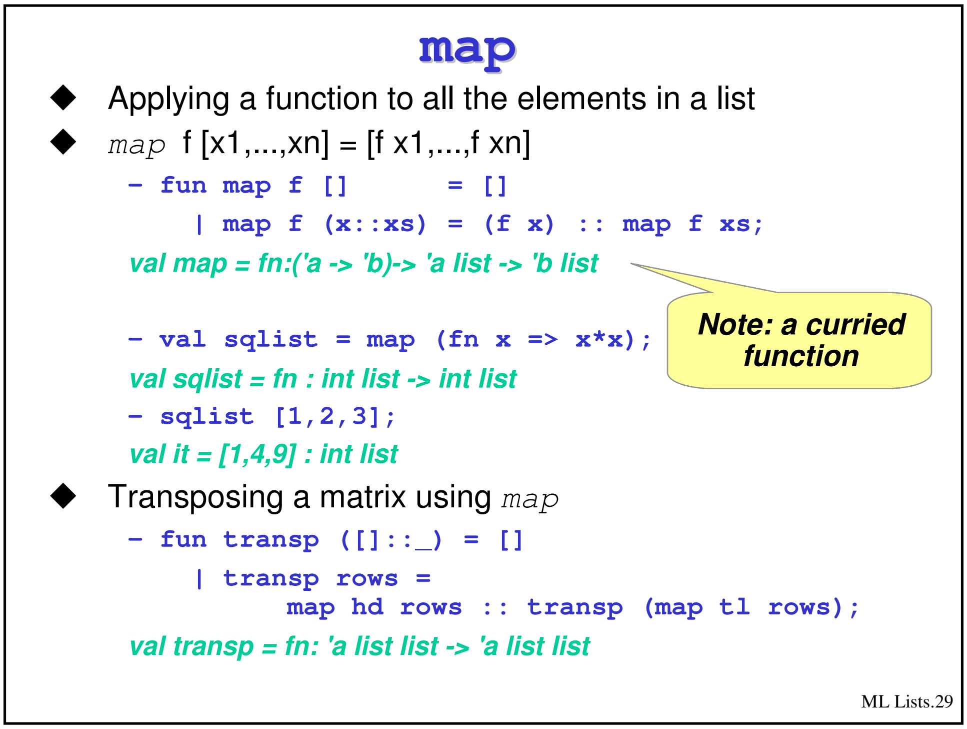
ML Lists.27

Built-in String functions

- In Standard ML, *string* is a primitive type, not a list of characters.

```
- explode "Technion";
val it = [#"T", #"e", #"c", #"h", #"n", #"i", #"o", #"n"]
   : char list
- implode it;
val it = "Technion" : string
- size(it) = length(explode(it));
val it = true : bool
```

ML Lists.28



map

- Applying a function to all the elements in a list
- *map* f [x1,...,xn] = [f x1,...,f xn]

```
- fun map f []      = []
    | map f (x::xs) = (f x) :: map f xs;
```

val map = fn:('a -> 'b)-> 'a list -> 'b list

Note: a curried function

```
- val sqlist = map (fn x => x*x);
```

val sqlist = fn : int list -> int list

```
- sqlist [1,2,3];
```

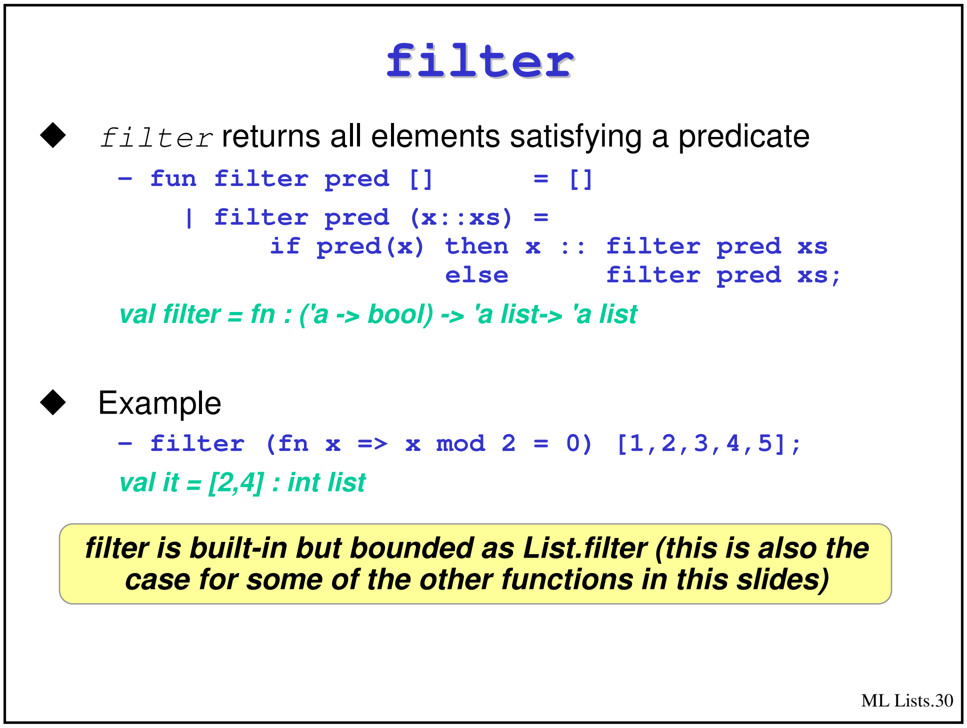
val it = [1,4,9] : int list

- Transposing a matrix using *map*

```
- fun transp ([]::_) = []
    | transp rows =
           map hd rows :: transp (map tl rows);
```

val transp = fn: 'a list list -> 'a list list

ML Lists.29



filter

- *filter* returns all elements satisfying a predicate

```
- fun filter pred []      = []
    | filter pred (x::xs) =
          if pred(x) then x :: filter pred xs
                     else      filter pred xs;
```

val filter = fn : ('a -> bool) -> 'a list-> 'a list

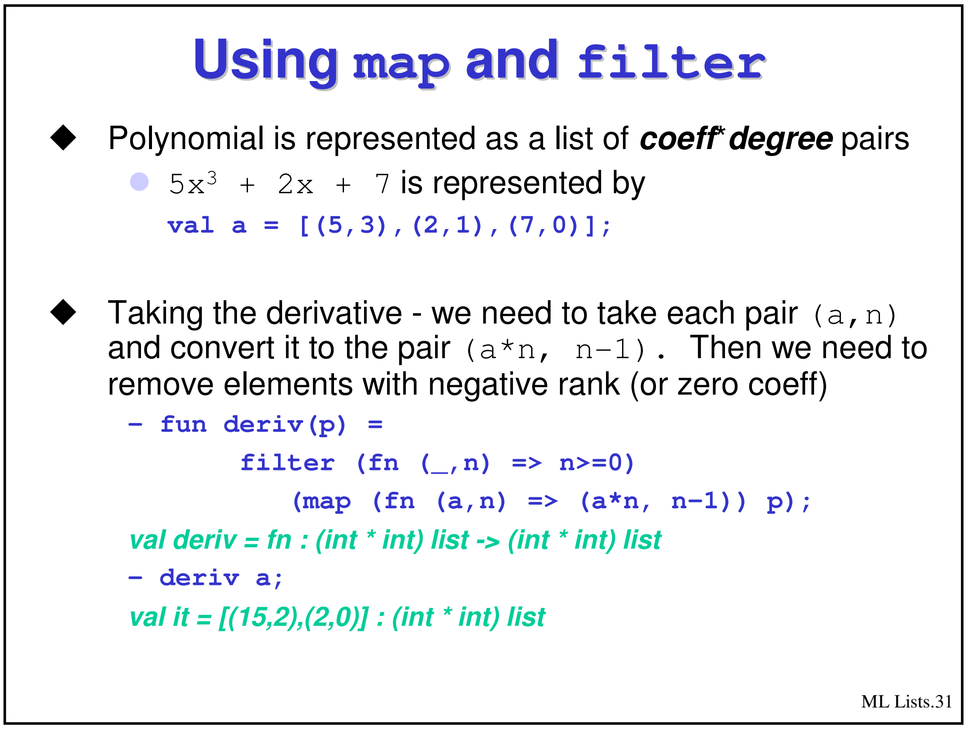
- Example

```
- filter (fn x => x mod 2 = 0) [1,2,3,4,5];
```

val it = [2,4] : int list

filter is built-in but bounded as List.filter (this is also the case for some of the other functions in this slides)

ML Lists.30



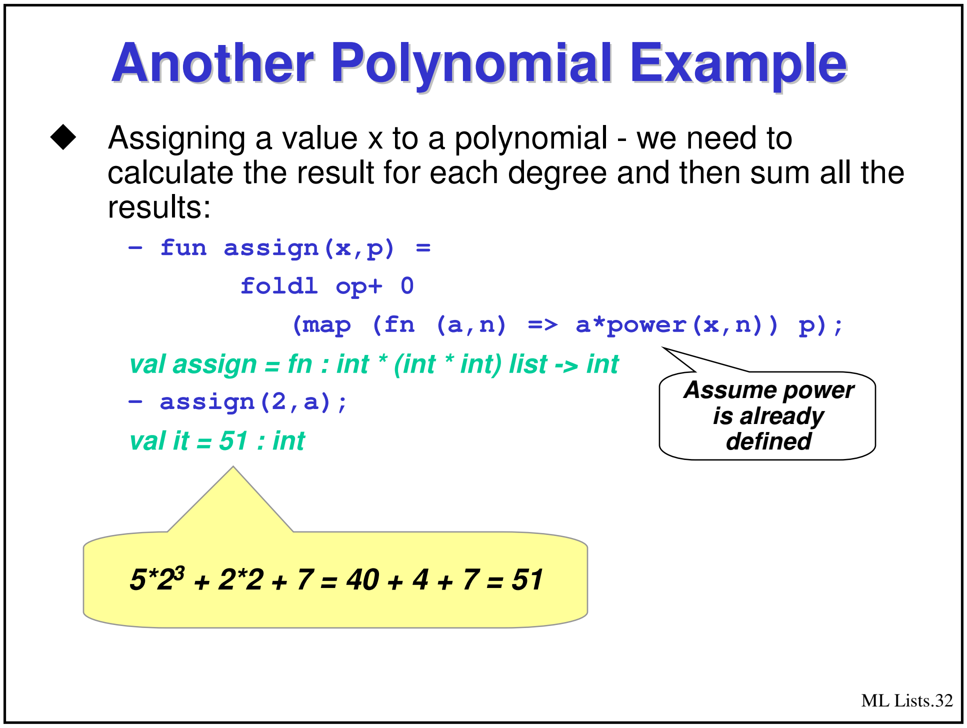
Using `map` and `filter`

- Polynomial is represented as a list of ***coeff*degree*** pairs
  - `5x`^{3}` + 2x + 7` is represented by

    ```
    val a = [(5,3),(2,1),(7,0)];
    ```

- Taking the derivative - we need to take each pair `(a,n)` and convert it to the pair `(a*n, n-1)`. Then we need to remove elements with negative rank (or zero coeff)

```
- fun deriv(p) =
       filter (fn (_,n) => n>=0)
          (map (fn (a,n) => (a*n, n-1)) p);
val deriv = fn : (int * int) list -> (int * int) list
- deriv a;
val it = [(15,2),(2,0)] : (int * int) list
```



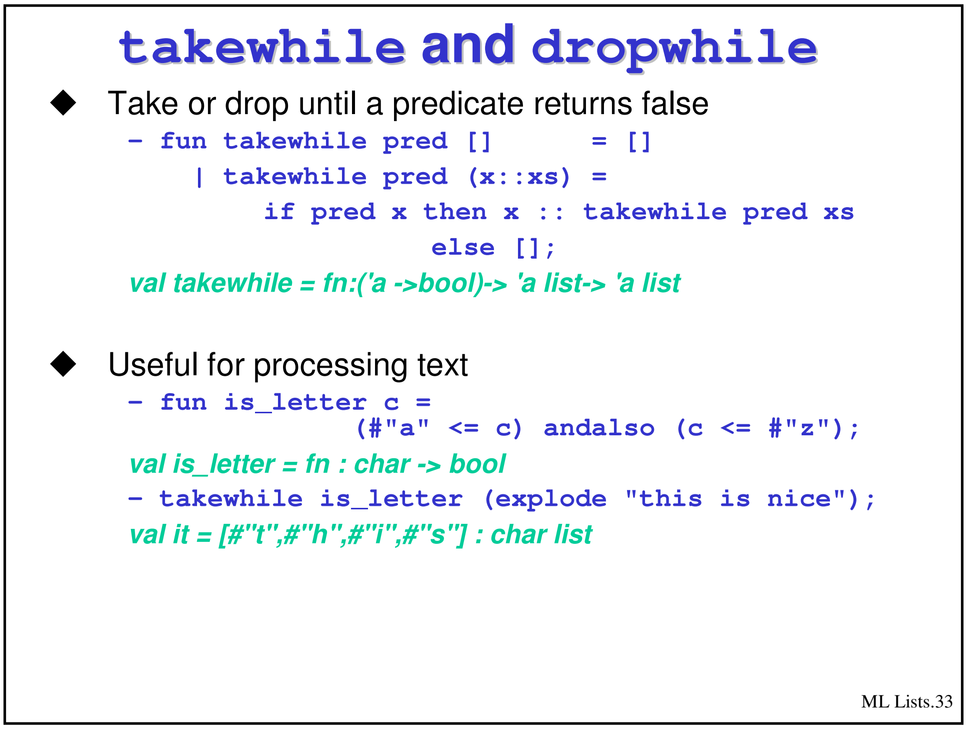
Another Polynomial Example

- Assigning a value x to a polynomial - we need to calculate the result for each degree and then sum all the results:

```
- fun assign(x,p) =
       foldl op+ 0
          (map (fn (a,n) => a*power(x,n)) p);
val assign = fn : int * (int * int) list -> int
- assign(2,a);
val it = 51 : int
```

Assume power is already defined

*5*2^3 + 2*2 + 7 = 40 + 4 + 7 = 51*



takewhile and dropwhile

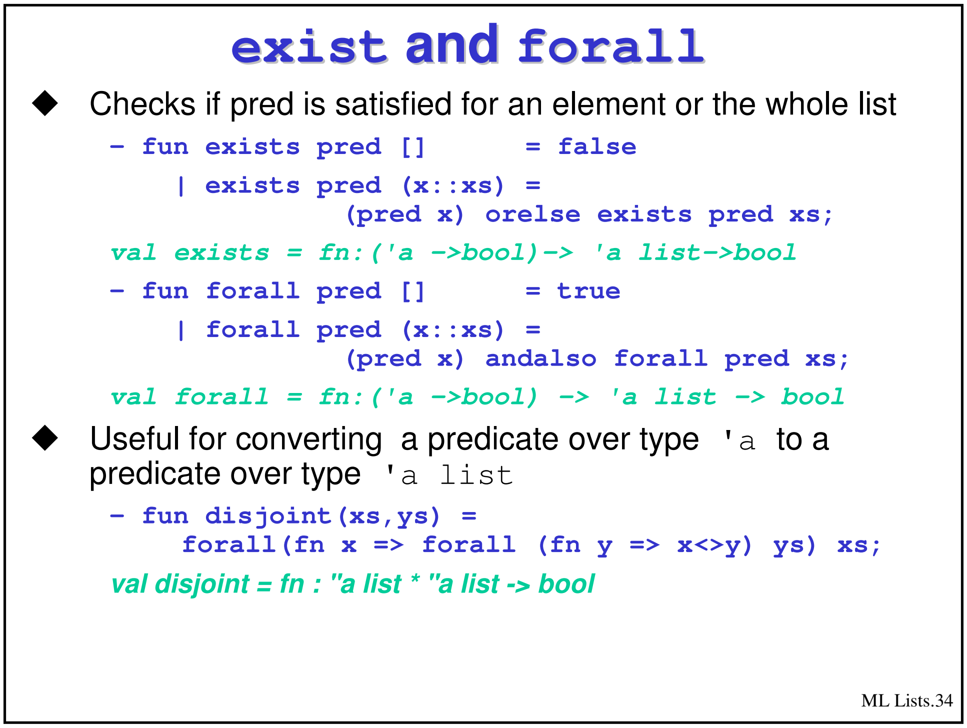
- Take or drop until a predicate returns false

```
- fun takewhile pred []       = []
    | takewhile pred (x::xs)  =
          if pred x then x :: takewhile pred xs
                     else [];
val takewhile = fn:('a ->bool)-> 'a list-> 'a list
```

- Useful for processing text

```
- fun is_letter c =
              (#"a" <= c) andalso (c <= #"z");
val is_letter = fn : char -> bool
- takewhile is_letter (explode "this is nice");
val it = [#"t",#"h",#"i",#"s"] : char list
```

ML Lists.33



exist and forall

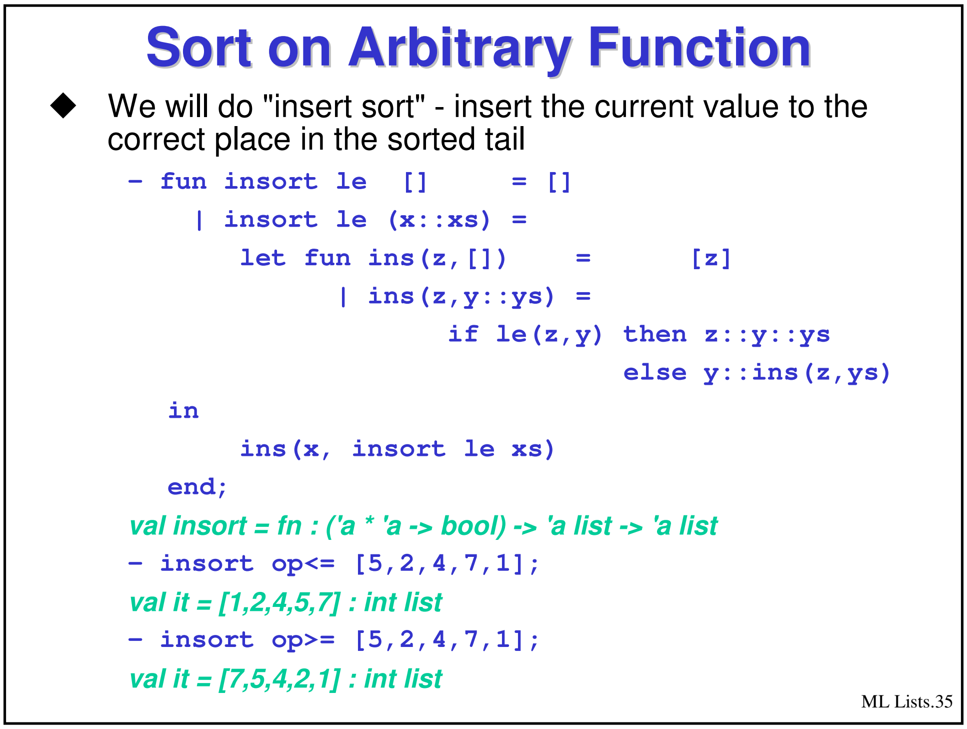
- Checks if pred is satisfied for an element or the whole list

```
- fun exists pred []       = false
    | exists pred (x::xs) =
              (pred x) orelse exists pred xs;
val exists = fn:('a ->bool)-> 'a list->bool
- fun forall pred []       = true
    | forall pred (x::xs) =
              (pred x) andalso forall pred xs;
val forall = fn:('a ->bool) -> 'a list -> bool
```

- Useful for converting a predicate over type `'a` to a predicate over type `'a list`

```
- fun disjoint(xs,ys) =
      forall(fn x => forall (fn y => x<>y) ys) xs;
val disjoint = fn : "a list * "a list -> bool
```

ML Lists.34

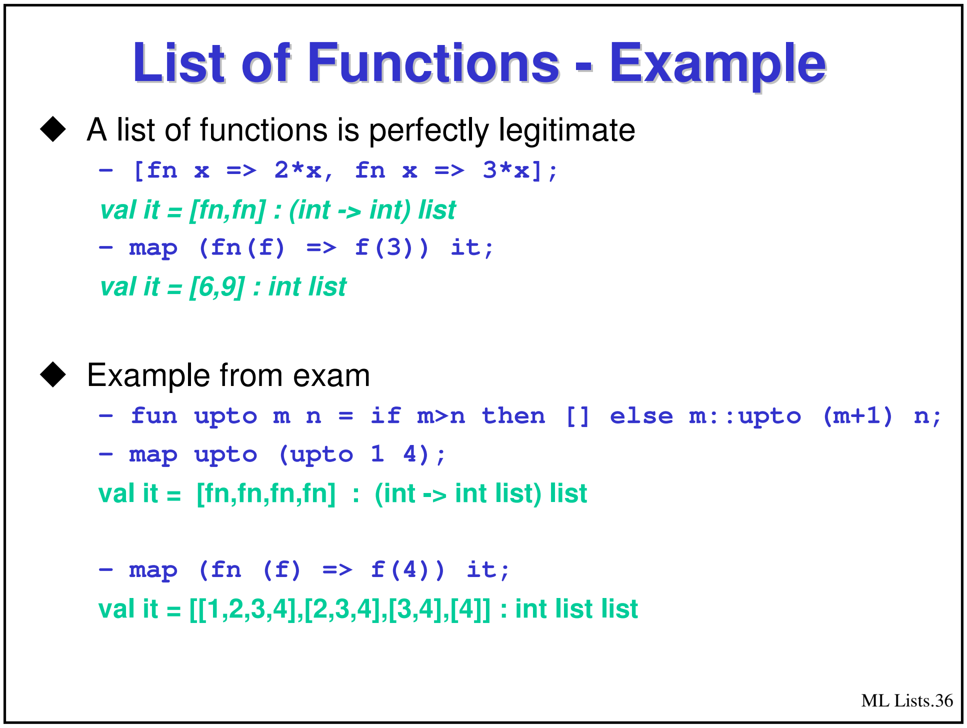


Sort on Arbitrary Function

- We will do "insert sort" - insert the current value to the correct place in the sorted tail

```
- fun insort le  []      = []
    | insort le (x::xs) =
       let fun ins(z,[])    =      [z]
             | ins(z,y::ys) =
                   if le(z,y) then z::y::ys
                              else y::ins(z,ys)
  in
       ins(x, insort le xs)
  end;
val insort = fn : ('a * 'a -> bool) -> 'a list -> 'a list
- insort op<= [5,2,4,7,1];
val it = [1,2,4,5,7] : int list
- insort op>= [5,2,4,7,1];
val it = [7,5,4,2,1] : int list
```

ML Lists.35



List of Functions - Example

- A list of functions is perfectly legitimate

```
- [fn x => 2*x, fn x => 3*x];
val it = [fn,fn] : (int -> int) list
- map (fn(f) => f(3)) it;
val it = [6,9] : int list
```

- Example from exam

```
- fun upto m n = if m>n then [] else m::upto (m+1) n;
- map upto (upto 1 4);
val it =  [fn,fn,fn,fn]  :  (int -> int list) list

- map (fn (f) => f(4)) it;
val it = [[1,2,3,4],[2,3,4],[3,4],[4]] : int list list
```

ML Lists.36