Daniel Leeds, 15-212 R02, September 5, 2007

Conventions:
Right associative: ->, ::, @
E.g.: 'a -> 'b -> 'c = 'a -> ('b -> 'c)
Left associative: orelse, andalso
E.g.: e_1 orelse e_2 andalso e_3 = (e_1 orelse e_2) andalso e_3

Binding strength: (in decreasing order)
list > * > ->
+,-,*,div > ::,@ > =,<
E.g.: int -> int * int = int -> (int * int)

Passing functions:
```
fun useful a [] = []      (* What does this do? *)
|   useful a (b::L) = (a b)::(useful a L)
```

Proofs:

```
fun rev [] = []
|   rev (h::L) = (rev L) @ [h]
```

```
fun rev2 [] acc = acc
|   rev2 (h::L) acc = rev2 L (h::acc)
```

Prove: (rev L) @ A = rev2 L A

```
fun ins x [] = [x]
  | ins x (y::L) = if (x<=y) then x::y::L else y::(ins x L)

fun sort [] = []
  | sort (x::L) = ins x (sort L)

fun listEq [] [] = true
  | listEq (x::xs) (y::ys) = (x=y) andalso (listEq xs ys)

LEMMA: ins x (ins y L) = ins y (ins x L)

By induction on length of list L.

Assume WLOG x < y

Base Case: L = []
  ins x (ins y []) =
    ins x [y]
    ins x (y::[])
    if (x<=y) then x::y::[] else ...
    x::y::[]
    x::([y])
    x::(ins y [])
    if (y<=x) then ... else x::(ins y [])
    ins y (x::[])
    ins y ([x])
    ins y (ins x [])

Inductive Hyp: Assume true for some list L of length n
Inductive Case: L'=z::L of length n+1
  ins x (ins y (z::L)) =
    ins x (if (y<=z) then y::z::L else z::(ins y L))

CASE 1: x < y <= z
    ins x (y::z::L)
    if (x<=y) then x::y::z::L else ...
    x::y::z::L
    x::(y::z::L)
    x::(ins y z::L)
    if (y<=x) then ... else x::(ins y z::L)
    ins y (x::z::L)
    ins y (if (x<=z) then x::z::L else ...)
    ins y (ins x (z::L))

CASE 2: x <= z < y
    ins x (z::(ins y L))
    if (x<=z) then x::z::(ins y L) else ...
    x::z::(ins y L)
    x::(z::(ins y L))
    x::(if (y<=z) then ... else z::(ins y L))
    x::(ins y z::L)
    if (y<=x) then ... else x::(ins y z::L)
    ins y (x::z::L)
    ins y (if (x<=z) then x::z::L else ...)
    ins y (ins x (z::L))

CASE 3: z < x < y
    ins x (z::(ins y L))
    if (x<=z) then ... else z::(ins x (ins y L))
    z::(ins x (ins y L))
    z::(ins y (ins x L))
    if (y<=z) then ... else z::(ins y (ins x L))
    ins y (z::(ins x L))
    ins y (if (x<=z) then ... else z::(ins x L))
    ins y (ins x (z::L))

The case where x>y is now trivial:
    ins y (ins x L) = ins x (ins y L)

Similarly x=y is easy:
    ins x (ins y L) = ins x (ins x L) = ins y (ins x L)
```