Lecture 11
Higher-Order Functions
(Part 2)
Composition as a combinator

\texttt{infix \ o}

\texttt{val \ o = fn (f,g) => fn x => f(g(x))}

\texttt{fun f \ o \ g = fn x => f(g(x))}

\texttt{fun (f \ o \ g) x = f(g(x))}
Combinators

• If we have functions of type \( \text{int} \rightarrow \text{int} \) we can “lift” integer operations from the space of integers to the space of these functions.
combinators

\[ f_1 \quad +\quad f_2 \quad *\quad \text{MIN} \]

Space of functions that return integers

\[ \text{Int.min} \]

Space of integers
Some infix combinators

\textbf{infixr} ++

\textbf{infixr} **

\textbf{fun} (f ++ g) x = f(x) + g(x)

\textbf{fun} (f ** g) x = f(x) * g(x)

What are the types of ++ and **?
Example usage

fun square x = x * x

fun double x = 2 * x

val quadratic = square ++ double
A prefix combinator

fun MIN (f, g) x = Int.min (f(x), g(x))

val lowest = MIN(square, double)
Staging Computation
fun f1 (x:int, y:int) : int =
  let
    val z: int = horrible(x)
  in
    z + y
  end

If it takes a long time for f1(5,10) to compute because of some horrible computation involving x, how long does it take to compute f1(5,7)?
Try with curry

```ml
fun f2 (x:int) (y:int) : int =
  let
    val z: int = horrible(x)
  in
    z + y
  end
```

If it takes a long time for \( f2 \ 5 \ 10 \) to compute because of some horrible computation involving \( x \), how long does it take to compute \( f2 \ 5 \ 7 \)?

Could we do better if we partially apply \( f2 \) as in \( \text{val} \ f2' = f2 \ 5 \) and then use \( f2' \ 10 \) and \( f2' \ 7 \)?
fun f3 (x:int) : int -> int =

let
  val z: int = horrible(x)

in

end
fun f3 (x:int) : int -> int =
  let
    val z: int = horrible(x)
  in
  fn y => z + y
end

If it takes a long time for \( f3 \ 5 \ 10 \) to compute because of some horrible computation involving \( x \), how long does it take \( f3 \ 5 \ 7 \)?

Could we do better if we partially apply \( f3 \) as in \( \textbf{val} \ f3' = f3 \ 5 \) and then use \( f3' \ 10 \) and \( f3' \ 7 \)?
Generalizing map and filter to datatypes
datatype 'a tree = Empty | Node of 'a tree * 'a * 'a tree

(* treemap : ('a -> 'b) -> 'a tree -> 'b tree
  REQUIRES: (maybe assume f is total)
  ENSURES: treemap f t returns a tree isomorphic to t in which each
  x of t has been replaced by f(x).
*)

fun treemap f Empty = Empty
  | treemap f (Node(left,x,right)) = Node(treemap f left, f x, treemap f right)

(* turn each integer of an int tree into a string: *)
(* stringify : int tree -> string tree *)

val stringify = treemap Int.toString
(* treefilter : ('a -> bool) -> 'a tree -> 'a tree
  REQUIRES: (maybe assume p is total)
  ENSURES: treefilter p t returns a tree consisting of all elements of t that satisfy p. *)

fun treefilter p Empty = Empty
| treefilter p (Node(left,x,right)) =
  if p(x) then ______________________________
  else ______________________________
(* treefilter : ('a -> bool) -> 'a tree -> 'a tree
    REQUIRES: (maybe assume p is total)
    ENSURES: treefilter p t returns a tree consisting of all elements of t that satisfy p. *)

fun treefilter p Empty = Empty
| treefilter p (Node(left,x,right)) = 
    if p(x) then Node(treefilter p left, x, treefilter p right) 
    else combine(treefilter p left, treefilter p right)
(*) combine : 'a tree * 'a tree -> 'a tree
    REQUIRES: true
    ENSURES: combine(t1,t2) returns a tree containing all the elements of t1
             and all the elements of t2 (with duplicates when such exist).

NOTE: The function finds the leftmost Empty of t1 and inserts t2 there.
*)

fun combine (Empty, t2) = t2
    | combine (t1, Empty) = t1  (* don't really need this; just a speedup *)
    | combine (Node(left, x, right), t2) = Node(combine(left,t2), x, right)