Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

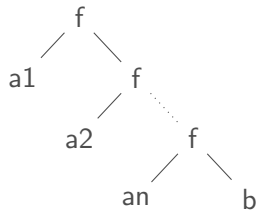
Week 4 - Sequence 5: Functions on Lists: Folding

Mapping and Folding

- Previous sequence: mapping a *unary* function over a list
- Mapping: All elements considered in isolation
- Folding: Combining all elements of a list using a *binary* operator
- Two different ways of folding: fold-left and fold-right

Folding Right

- Type de `List.fold_right`: `('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`
- Computing `List.fold_right f [a1; a2; ... an] b`:



Folding Right I

```
let rec fold_right f l b = match l with
  | [] -> b
  | h::r -> f h (fold_right f r b);;
# val fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b =
  <fun>

fold_right (fun x y -> x::y) [1;2;3;4] [];;
# - : int list = [1; 2; 3; 4]

fold_right (+) [1;2;3;4] 0;;
# - : int = 10

fold_right ( * ) [1;2;3;4] 1;;
# - : int = 24
```

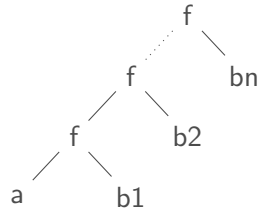
Folding Right II

```
let concat = fold_right (fun x y -> x::y);;
# val concat : '_a list -> '_a list -> '_a list = <fun>

concat [1;2;3;4] [5;6;7;8];;
# - : int list = [1; 2; 3; 4; 5; 6; 7; 8]
```

Folding Left

- Type de `List.fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
- Computing `List.fold_left f a [b1; b2; ... bn]`



Folding left I

```
(* defined in library as List.fold_left *)
let rec fold_left f b = function
  | [] -> b
  | h::r -> fold_left f (f b h) r;;
# val fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a =
  <fun>

fold_left (+) 0 [1;2;3;4];;
# - : int = 10

let reverse = fold_left (fun x y -> y::x) [];;
# val reverse : '_a list -> '_a list = <fun>

reverse [1;2;3;4;5];;
# - : int list = [5; 4; 3; 2; 1]
```

Case Study: Inner Product of Integer Vectors

- Inner Product:

$$[x_1; x_2; x_3] * [y_1; y_2; y_3] = [x_1 * y_1 + x_2 * y_2 + x_3 * y_3]$$

- Computation : first compute list of pair-wise products, then sum up.

Example: Inner Product of Integer Vectors I

```
let product v1 v2 =
  List.fold_left (+) 0 (List.map2 ( * ) v1 v2);;
# val product : int list -> int list -> int = <fun>

product [2;4;6] [1;3;5];;
# - : int = 44
```

Example: Counting Elements of a List I

```
(* count elements of l satisfying p *)
let countif p l = List.fold_left
  (fun counter element -> if p element then counter+1 else counter)
  0
  l
;;
# val countif : ('a -> bool) -> 'a list -> int = <fun>

countif (function x -> x>0) [3;-17;42;-73;-256];;
# - : int = 2
```

To Know More

The OCaml Manual:

- The standard library
  - Module `List`