Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 4 - Sequence 4: Functions on Lists: Maps

The library module `List`

- The `List` library contains many useful functions.
- Use after **`open`** `List`, or with pointed notation (like `List.hd`).
- More about modules in Week 6.
- Polymorphism gives us generality.

Mapping I

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

map (function x -> x*x) [1;2;3;4;5];;
# - : int list = [1; 4; 9; 16; 25]
```

Mapping over Two Lists I

```
(* defined in library as List.map2 *)
let rec map2 f l1 l2 = match (l1,l2) with
  | [],[] -> []
  | h1::r1,h2::r2 -> (f h1 h2)::(map2 f r1 r2)
  | _ -> raise (Invalid_argument "List.map2");;
# val map2 :
  ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list = <fun>

map2 (fun x y -> x+y) [1;2;3] [10;20;30];;
# - : int list = [11; 22; 33]
```

Example: Integer Vectors and Matrices

- Represent a row-vector of integers as a list.
- Represent a matrix of integers as a list of row-vectors.
- Turning an infix operator into a function: `(+)`, `(/)`, ...
- Special case multiplication: `( * )`

Sum of Two Vectors I

```
let vsum = List.map2 (+);;
# val vsum : int list -> int list -> int list = <fun>

vsum [1;2;3] [10;20;30];;
# - : int list = [11; 22; 33]
```

Sum of Two Matrices I

```
let msum = List.map2 (List.map2 (+));;
# val msum : int list list -> int list list -> int list list =
  <fun>

msum [ [1;2]; [3;4]] [[10;20]; [30;40]];;
# - : int list list = [[11; 22]; [33; 44]]
```

Example: Lists and Sublists

- Sublist of *l*: obtained from *l* by removing some of its elements
- Example: `[2;4]` sublist of `[1;2;3;4]`
- Task: compute the set of sublists of a given list
- Type: `'a list -> 'a list list`
- `sublists [] = [[]]`
- if `l=h::r`, then any sublist of `l`
  - either is some sublist of `r`,
  - or obtained from some sublist of `r` by putting `h` in front

Computing Sublists I

```
let rec sublists = function
  | [] -> [ [] ]
  | h::r ->
      let rp = sublists r in
        rp@(List.map (function l -> h::l) rp);;
# val sublists : 'a list -> 'a list list = <fun>

sublists [1;2;3];;
# - : int list list =
[[]; [3]; [2]; [2; 3]; [1]; [1; 3]; [1; 2]; [1; 2; 3]]
```

To Know More

The OCaml Manual:

- The standard library
  - Module `List`