FILTER

Titolo nota 22/10/2015

Data una lista e una proprietà sugli elementi della lista, costruire la lista dei soli elementi che soddisfano la proprietà.

[-1; 3; 5; 8; -7] la proprietà "maggiore di 0", filter restituisce la lista

[3;5;8]

```
let rec filter p l = match l with
      []      ->   []
  | x::xs  when  (p x)      ->   x::(filter p xs)
  | x::xs  when  not (p x)  ->   filter p xs;;
```

filter : ('a -> bool) -> 'a list -> 'a list = <fun>

('a -> bool): tipo p; 'a list: tipo l; 'a list: ris.

Titolo nota 22/10/2015

Esempio

```
let foo l = let p x = x mod 3 = 0 in
            filter p l ;;
```

foo : int 'a list ⟶ int list = <fun>

(int = tipo l; int list = tipo ris)

filter : ('a → bool) → 'a list → 'a list

p : int → bool

'a e' int

foo [3;8;6]

= { def. di foo }

filter p [3;8;6]

= { 2° patt., x=3, xs=[8;6], 3 mod 3 = 0 }

3 :: (filter p [8;6])

= { 3° pattern, x=8, xs=[6], 8 mod 3 ≠ 0 }

3 :: (filter p [6])

= { 2° pattern, x=6, xs=[], 6 mod 3 = 0 }

3 :: 6 :: (filter p [])

= { 1° pattern di filter }

3 :: 6 :: []

=

[3;6]

Titolo nota 22/10/2015

Scrivere una funzione che, data una lista di interi e un intero x, restituisce una coppia di liste (l1, l2) in cui l1 contiene tutti gli elementi della lista maggiori di x, e l2 contiene tutti gli altri elementi. Non vogliamo usare RICORSIONE ESPLICITA.

```
let fie l x = let magx y = y > x
              in let nonmag y = y <= x
in
    filter magx l , filter nonmag l
```

fie : int list → int → int 'a list * int 'b list = <fun>

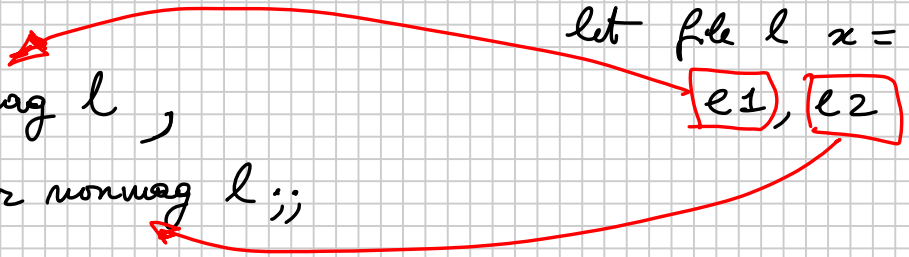
(int list = l, int = x, int 'a list * int 'b list = ris.)

magx: int → bool (int = y)

nonmag : int → bool (int = y, bool = ris.)

Titolo nota 08/10/2015

```
let fie l x =
    let mag y = y>x in filter mag l ,
    let nonmag y = y<=x in filter nonmag l ;;
```



FOLDR

Titolo nota 08/10/2015

① $\sum_{x_i \in l} x_i = x_1 + \sum_{x_i \in tl\ l} x_i$ se l non vuota

② $\sum_{x_i \in []} x_i = \boxed{\emptyset}$

$\sum_{x_i \in l} x_i = x_1 \boxed{+} \underbrace{(x_2 + \dots (x_n + \emptyset) \dots)}_{\sum_{x_i \in tl\ l} x_i}$

$\prod_{x_i \in l} x_i = x_1 \boxed{*} \prod_{x_i \in tl\ l} x_i$

$\prod_{x_i \in []} x_i = \boxed{1}$

data una lista l
$[x_1; x_2; \dots; x_n]$
$x_i \in l$

$\bigwedge_{x_i \in l} x_i = x_1 \boxed{\wedge} \bigwedge_{x_i \in tl\ l} x_i$

$\bigwedge_{x_i \in []} x_i = \boxed{true}$

FOLDR

Titolo nota 08/10/2015

```
let rec foldr f a l = match l with
        []  -> a
      | x::xs -> f x (foldr f a xs) ;;
```

```
match exp with
   p1 -> e1
|  p2 -> e2
```
↓ stesso tipo

foldr : ('a → 'b → 'b) → 'b → 'a list → 'b

('a → 'b → 'b) = f, 'b = a, 'a list = l, 'b = ris

```
# let sommatoria l = let f x y = x+y in foldr f 0 l ;;
# let produttoria l = let f x y = x*y in foldr f 1 l ;;
# let andatoria l = let f x y = x&y in foldr f true l ;;
```

Calcolare la lunghezza di una lista senza ricorsione esplicita

Titolo nota 08/10/2015

$\sum_{x_i \in l} 1$ $\qquad$ $\sum_{x_i \in []} 1 = \phi$

```
let lung l = let f x y = 1 + y
             in
                foldr f ? l
```

Quando "progettiamo" la funzione binaria f da utilizzare dentro foldr

```
f x y
```

x sta per un generico elemento della lista

y sta per il risultato parziale dell'op. che state definendo su tutta la lista che segue x

$[\ldots; x; x_1; \ldots; x_k]$

x $\qquad$ y

f deve aggiungere il contributo di x al risultato parziale

let g l = let f x y = ... in foldr f a l;;

g [x_1; ....; x_n]

= { def. di g }

foldr f a [x_1; ....; x_n]

= { 2° pattern di foldr }

f x_1 (foldr f a [x_2; ....; x_n])

g [x_2; ....; x_n]

el. della lista

risultato di g su tutto ciò che nella lista originale segue il primo argomento di f

Titolo nota 08/10/2015

Data una lista di interi calcolare la differenza tra il numero di elementi >0 nella lista e il numero di elementi negativi o nulli nella lista.

foo $[-1; 3; 4; 5; -8; -7; -10; \emptyset] = -2$

```
let foo l = let f x y = if x>0 then 1+y else y-1
            in foldr f ∅ l
```

MAP attraverso foldr

Titolo nota 08/10/2015

```
let rec map f l = match l with
    []  ->  []
  | x::xs -> (f x) :: (map f xs);;
```

```
let rec filter p l = match l with
    []  ->  []
  | x::xs when p x -> x :: (filter p xs)
  | x::xs when not (p x) -> filter p xs;;
```

```
let mapp f l =
    let g x y = (f x)::y
    in
        foldr g [] l
```

```
let ffilter p l =
    let f x y = if p x then x::y
                       else y
    in
        foldr f [] l
```

Calcolare il massimo valore in una lista (assumiamo che la lista contenga valori positivi)

Titolo nota 08/10/2015

```
let max l = let f x y = if x>y then x else y
            in foldr f ∅ l
```

Tipo di max

foldr : ('a → 'b → 'b) → 'b → 'a list → 'b

max : int list → int

(int list = l, int = ris)

max [2;10;3]

= foldr f ∅ [2;10;3]

= f 2 (foldr f ∅ [10;3])

= f 2 (f 10 (foldr f ∅ [3]))

= f 2 (f 10 (f 3 (foldr f ∅ [])))

= f 2 (f 10 (f 3 ∅))

= f 2 (f 10 3)

= f 2 10

= 10

Calcolare il valore massimo in una lista NON VUOTA

Titolo nota 08/10/2015

```
let max l = let f x y = if x>y then x else y
            in
            ~~foldr f ? l~~        match l with
                                   x::xs ⟶ foldr f x xs
```

```
   max [8;-1]
=  { x=8, xs=[-1] }
   foldr f 8 [-1]
=  f (-1) (foldr f 8 [])
=  f (-1) 8
=  8
```

```
   max [3; 10; 5]
=  { x=3; xs=[10;5] }
   ⋮
=  f 10 (f 5 (foldr f 3 []))
=  f 10 (f 5 3)
=  (f 10 5)  =  10
```

Calcolare minimo e massimo di una lista

Titolo nota 08/10/2015

```
let minmax l = let f x (m1, m2) = if x < m1 then (x, m2) else
                                  if (x > m2) then (m1, x)
                                  else (m1, m2)
               in match l with
                  x::xs -> foldr f (x,x) xs
```

minmax : 'a list → 'a * 'a = <fun>

foldr : ('a → 'b → 'b) → 'b → 'a list → 'b

('a * 'a) ('a * a) ('a * 'a) ('a * 'a)