foldr f a [x1; x2; ... ; xn] =

f x1 (f x2 (... (f xn a) ...))

foldr f a [x1; x2; x3] =

f x1 (f x2 (f x3 a))

valore della lista

risultato della foldr sugli elementi che seguono

Somma degli elementi di una lista utilizzando foldr

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

foldr : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b

('a -> 'b -> 'b) : tipo f

'b : tipo a

'a list : tipo l

'b : tipo xs

```
let somma l =
    let f x y = x + y
    in foldr f 0 l ;;
```

Definizione della funzione filter

filter p l conserva nel risultato tutti gli elementi di l in cui il predicato p è vero.

usando foldr

```
let filter p l =
    let f x y = if p x then x::y
                       else y
    in foldr f [] l ;;
filter : ('a -> bool) -> 'a list -> 'a list
```

foldr f a [x1; x2; x3] =

f x1 (f x2 (f x3 a))

Conviene usare le foldr per operazioni che sono definite più semplicemente partendo dal FONDO DELLA LISTA

map f l dà come risultato la lista l ai cui elementi è stata applicata la funzione f

map f $[x_1; x_2; x_3]$ =

$[f\,x_1; f\,x_2; f\,x_3]$

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

map : ('a -> 'b) -> 'a list -> 'b list

map con foldr

```
let map f l =
    let g x y = f x :: y
    in foldr g [] l ;;
```

map : ('a -> 'b) -> 'a lst -> 'b lst

---

map incr [3;4;5]

g x y = incr x :: y

g 3 (g 4 (g 5 []))

g 5 [] = [6]

g 4 [6] = [5;6]

g 3 [5;6] = [4;5;6]

Date una lista dividerla in due liste (l1, l2) tali che l1 contiene tutti gli elementi di l >=0, l2 tutti quelli < 0.

```
let rec split l = match l with
    [] -> ([], [])
  | x::xs -> let (l1, l2) = split xs
             in
             if x>=0 then (x::l1, l2)
                     else (l1, x::l2);;
split : int list -> int list * int list
```

Split usando foldr

```
let split l =
    let f x y = match y with
                (l1, l2) -> ....
    in foldr f ([],[]) l ;;
```

OK

```
let f x (l1, l2) =
    if x >= 0 then (x :: l1, l2)
              else (l1, x :: l2)
```

minmax usando foldr

non è definita su liste vuote

```
let minmax l =
    let f x y = .....
    in
    foldr f .... l;;
```

non posso mettere il valore neutro su liste vuote!!

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

mercoledì 5 dicembre 2018 12:13

date una lista di coppie di interi vogliamo la lista dei valori ottenuta sommando i valori di ogni coppia

es: $[(3,5);(4,7);(-2,2)] \rightarrow [8;11;0]$

```
let rec sum l = match l with
    [] -> []
  | (n,m)::xs -> n+m :: sum xs ;;
```

sum : int * int list -> int list

Con foldr:

```
let sum l =
    let f (n,m) y = n+m :: y
    in foldr f [] l ;;
```

foldr f [] [(3,4);(5,6)] =

$f\ (3,4)\ \underbrace{\left(f\ (5,6)\ []\right)}_{11 :: []} = \underbrace{\quad\quad\quad\quad\quad\quad\quad\quad}_{[11]}$

$7 :: [11] \quad = \quad [7; 11]$

Cancellare gli ultimi n elementi di una lista

conc n l

es:

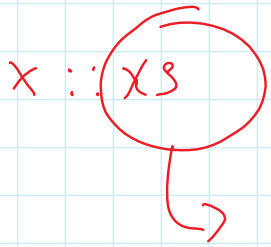
conc 2 [3;4;5] = [3]

conc ~~0~~ [3;4;5] = [3;4;5]

conc 7 [3;4;5] = []

```
let rec len l = match l with
    [] -> 0
  | x::xs -> 1 + len xs;;
len : 'a list -> int
```

```
let rec conc n l = match l with
    [] -> []
  | x::xs -> if len l <= n then []
             else x:: conc n xs;;
conc : int -> 'a list -> 'a list
```



applico la funzione ricorsivamente su xs.
x va cancellato dal risultato oppure no???

La chiamata ricorsiva su xs mi deve dare una lista come risultato, ma anche quanti elementi sono stati cancellati!!

```
let rec concm n l = match l with
    [] -> ([], ∅)
  | x::xs -> let (l, m) = concm n xs
             in if m < n
                then (l, m+1)
                else (x::l, m) ;;
```

concm 2 [3;4;5]
= {def concm, 2°p}
let (l, m) = ([], 2)
in ...
([3], 2)

concm 2 [5]
= {def 2°p}
let (l, m) = ([], ∅)
in ...

concm 2 [4;5]
= {def concm, 2°p}
let (l, m) = ([], 1)
in ...
=
([], 2)

concm 2 []
= {def concm, 1°p}
([], ∅)

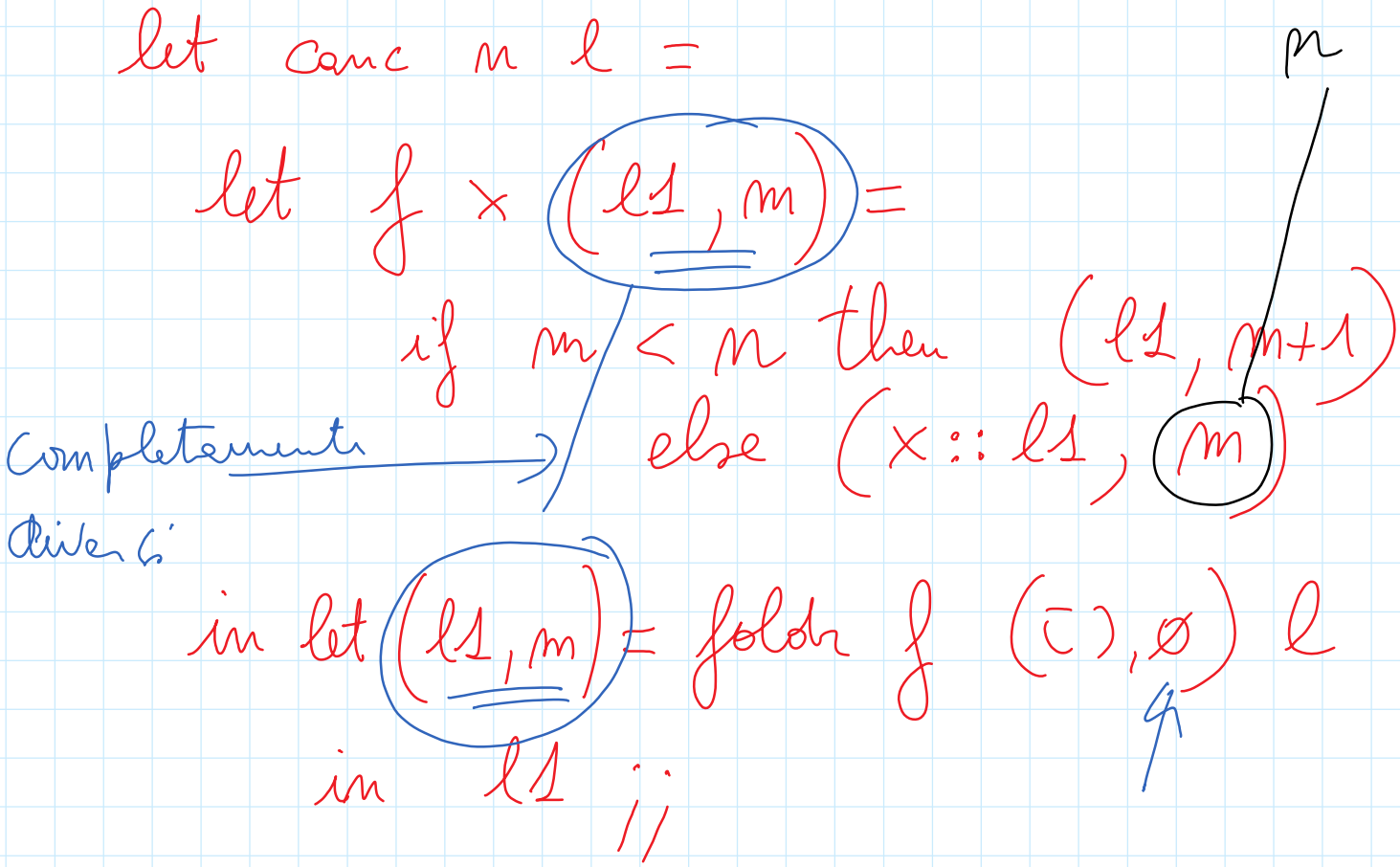
let (l, n) = ([], x)
in . . . .

= ([], 1)

conta : int -> 'a list -> 'a list * int

```
let conc n l =
    let rec concm n l = .....
    in let (l1,m) = concm n l
    in l1 ;;
```

conc : int → 'a list → 'a list


let conc n l =
let f x (l1, m) =
if m < n then (l1, m+1)
else (x :: l1, m)
n
Completamente diversi
in let (l1, m) = foldr f (□, ∅) l
in l1 ;;

Cancellare gli elementi di una lista di interi dopo l'ultima occorrenza del valore $\emptyset$.

$[3; \emptyset; -7; \emptyset; 4] \longrightarrow [3; \emptyset; -7; \emptyset]$

Se $\emptyset$ non compare nella lista bisogna cancellare tutti gli elementi.

$[3; 4; 5] \longrightarrow []$

foldr

f x y

y ci deve dare come risultato una lista ma anche l'indicazione se abbiamo incontrato un valore $\emptyset$ tra gli elementi che seguono x

(lista, valore di verità)

true se abbiamo incontrato $\emptyset$

false altrimenti.