

FUNZIONI DI ORDINE SUPERIORE

• $\text{map} : ('a \rightarrow 'b) \rightarrow 'a \text{ list} \rightarrow 'b \text{ list}$

$$\text{map } f [x_1; \dots; x_n] = [f x_1; \dots; f x_n]$$

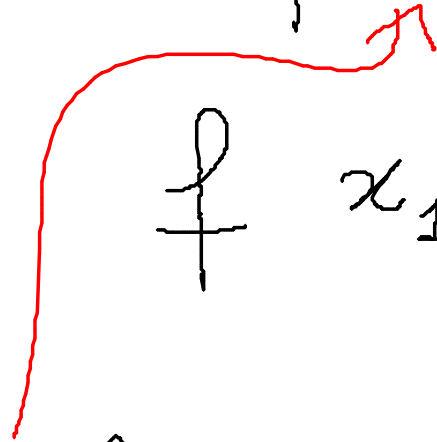
• $\text{filter} : ('a \rightarrow \text{bool}) \rightarrow 'a \text{ list} \rightarrow 'a \text{ list}$

che elimina da una lista tutti gli elementi che non soddisfano il predicato primo argomento

• foldr : operatore di fold

foldr f a $[x_1; \dots; x_n] =$

f x_1 (f x_2 (\dots (f x_n a) \dots))



è il risultato di foldr quando la lista è vuota
(elemento neutro)

f è una funzione BINARIA (a due argomenti)

CALCOLARE LA SOMMA DEGLI ELEMENTI DI UNA LISTA

$$\sum_{i=1}^m x_i$$

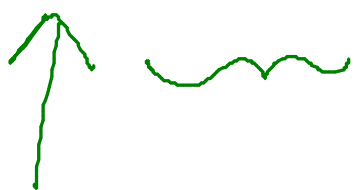
$$[x_1; \dots; x_m]$$

$$x_1 + \sum_{i=2}^m x_i$$

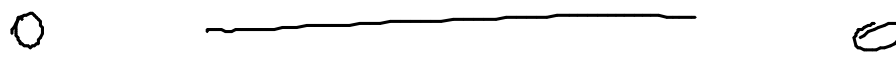
x_1 risultato parziale del calcolo sulla lista $[x_2; \dots; x_m]$

↳ si aggiunge al risultato parziale il contributo di x_1

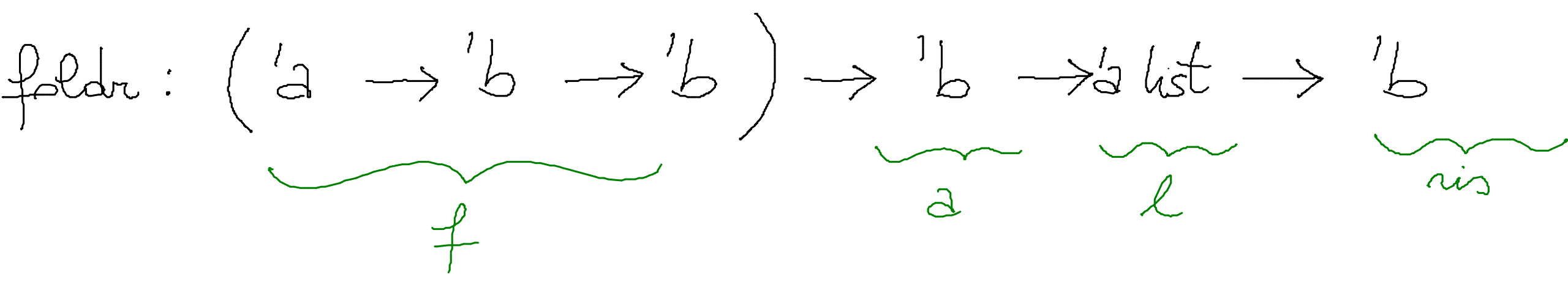
$$+ x_1 \left(+ x_2 \dots \left(+ x_m \emptyset \right) \dots \right)$$



$$\prod_{i=1}^n x_i = * x_1 (* x_2 \dots (* x_n 1) \dots)$$



let rec foldr f a l = match l with



$\sum_{i=1}^n x_i$ mediante foldr

let sum x y = x + y ;;
sum : int → int → int

let sum = prefix + ;;

let sumlist l = foldr sum \emptyset l ;;

sumlist : int list → int
l res.

foldr : ('a → 'b → 'b) → 'b → 'a list → 'b
sum : int → int → int =

sumlist [2; -1; 4]

= { def. di sumlist }

foldr sum \emptyset [2; -1; 4]

= { 2° pott. di foldr }

sum 2 (foldr sum \emptyset [-1; 4])

= { 2° pott. di foldr }

sum 2 (sum -1 (foldr sum \emptyset [4]))

= { 2° pott. di foldr }

sum 2 (sum -1 (sum 4 (foldr sum \emptyset [])))

= { 1° pott. di foldr }

sum 2 (sum -1 (sum 4 \emptyset))

let rec foldr f a l =

match l with

[] \rightarrow a |

x::xs \rightarrow f x (foldr f a xs);;

= { def. di sum }

2 + (-1) + (4 + 0)

= 5

let prod x y = x * y ;;
prod : int → int → int

let prodlst l = foldr prod 1 l ;;
prodlst : int list → int

let forall p l = $\underbrace{\text{let } f \ x \ y = (p \ x) \ \& \ y \ \text{in}}_{\text{foldr } f \ \text{true } l}$

forall p [x₁; ...; x_n] =

f x₁ (f x₂ (... (f x_n true) ...))
= (p x₁) & (p x₂) & ... & (p x_n) & true

let exists p l = let f x y = (p x) or y
in foldr f false l ;;

$(p x_1) \vee (p x_2) \vee \dots \vee (p x_n)$

COME SI RISOLVE UN PROBLEMA UTILIZZANDO FOLDR

$\text{let foo } l = \text{let } f \ x \ y = \dots$
 in
 $\text{foldr } f \ ? \ l$

un generico elemento di l
 risultato di foo se applicata alla lista che segue x in l

in modo che $\text{foo } l$ calcoli il numero di elementi maggiori di \emptyset di una lista di interi

$$\begin{aligned}
 \text{foo } [x_1; \dots; x_n] &= \text{foldr } f \ ? \ [x_1; \dots; x_n] \\
 &= f \ x_1 \left(\dots \left(f \ x_i \left(f \ x_{i+1} \dots \left(f \ x_n \ ? \right) \dots \right) \right) \right)
 \end{aligned}$$

$\text{foo } [x_{i+1}; \dots; x_n] = \text{foldr } f \ ? \ [x_{i+1}; \dots; x_n]$

let foo l = let f x y = if x > 0
then 1 + y
else y

in

foldr f 0 l

→ numero di elementi maggiori di ϕ nella
porzione di l che segue x

f x y deve aggiungere il contributo di
x al risultato

$f x y =$ if $x > 0$ then $1+y$
else y

$$f_{\infty} [-1; 2]$$

$$= \text{foldr } f \ \emptyset \ [-1; 2]$$

$$= f \ (-1) \ (\text{foldr } f \ \emptyset \ [2])$$

$$= f \ (-1) \ (f \ 2 \ (\text{foldr } f \ \emptyset \ []))$$

$=$ $\{1^{\circ}$ patt. di foldr $\}$ if $2 > 0$ then $1 + \emptyset$ else \emptyset

$$f \ (-1) \ (f \ 2 \ \emptyset)$$

if $(-1 > 0)$ then $1 + 1$ else 1

$$= f \ (-1) \ \underline{1}$$

$$= \boxed{1}$$

CALCOLARE IL NUMERO DI ELEMENTI e LA LORO SOMMA IN UNA LISTA

$$g [x_1; \dots; x_m] = (n, \sum_{i=1}^n x_i)$$

let rec g l = match l with
[] \rightarrow (0, 0)

se calcolo $g xs = (m, s)$

| x :: xs \rightarrow let $(m, s) = g xs$ in
(m+1, s+x) ;;

$$\text{let } g \text{ l} = \text{let } f \ x \ (m, s) = (m+1, s+\alpha)$$

in
foldr f (0,0) l

$g \ [1; 10; 5]$ $= \text{foldr } f \ (\emptyset, \emptyset) \ [1; 10; 5]$ $= \{2^\circ \text{ p. di foldr}\}$ $f \ 1 \ (\text{foldr } f \ (\emptyset, \emptyset) \ [10; 5])$ $= \{2^\circ \text{ p. di foldr}\}$ $f \ 1 \ (f \ 10 \ (\text{foldr } f \ (\emptyset, \emptyset) \ [5]))$	$= \{2^\circ \text{ p. di foldr}\}$ $f \ 1 \ (f \ 10 \ (f \ 5 \ (\text{foldr } f \ (\emptyset, \emptyset) \ [])))$ $= \{1^\circ \text{ p. di foldr}\}$ $f \ 1 \ (f \ 10 \ (f \ 5 \ (\emptyset, \emptyset)))$ $= f \ 1 \ (f \ 10 \ (1, 5))$ $f \ 1 \ (2, 15)$ $= (3, 16)$
---	--

Se non avessimo utilizzato il p.m. per rappresentare il risultato della chiamata ricorsiva nella prima def. di g , ovvero il secondo argomento di f nella def. di g mediante `foldr`

let rec $g\ l = \text{match } l \text{ with}$
 $[\] \rightarrow (\emptyset, \emptyset) \mid$

$x :: xs \rightarrow \text{let } (m, s) = g\ xs$
 $\text{in } (m+1, s+x)$

let first $(x, y) = x$;
 let second $(x, y) = y$;

$x :: xs \rightarrow (1 + \text{first } (g\ xs),$
 $x + \text{second } (g\ xs))$

Domanda : voglio definire f attraverso g, h , come
funzioni locali.

let $f \dots =$ let $\underline{g} \dots = \dots$
in
let $\underline{h} \dots = \dots$
in
 \dots

let $f \dots =$ let $g \dots = \dots$
and
 $h \dots = \dots$
in
 \dots

Portare, se c'è, l'ultimo elemento di una lista in testa alla lista

foo [10; 5; 6] = [6; 10; 5]

let foo l = let f [x] y = w::wS → w::x::wS

se y è della forma
w::wS → w::x::wS



y è xS in cui l'ultimo elemento è stato portato in testa

in ... foldr ...

let foo l = let f ³⁰ x y = match y with
[] → [x]

| w :: ws → w :: x :: ws

in foldr f [] l ;;

<p>= foo [10; 20; 30]</p> <p>= foldr f [] [10; 20; 30]</p> <p>= f 10 (foldr f [] [20; 30])</p> <p>= f 10 (f 20 (foldr f [] [30]))</p>	<p>= f 10 (f 20 (f 30 (foldr f [] [])))</p> <p>= f 10 (f 20 (f 30 []))</p> <p>= f 10 (f 20 [30])</p> <p>= f 10 30 :: 20 :: [] =</p> <p>= 30 :: 10 :: 20 :: []</p>
---	---

MAP e FILTER attraverso foldr.

let rec map f l = match l with

[] → []

| x :: xs → (f x) :: (map f xs) ;;

let mapp f l = let g x y = (f x) :: y

generico elemento
di l

risultato parziale
cioè map f xs dove
xs è tutto ciò che segue
x in l

in

foldr g [] l

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 filter p l = let f x y = if (p x) then x :: y
else y

in

foldr f [] l