CAML

lunedì 3 dicembre 2018 09:15

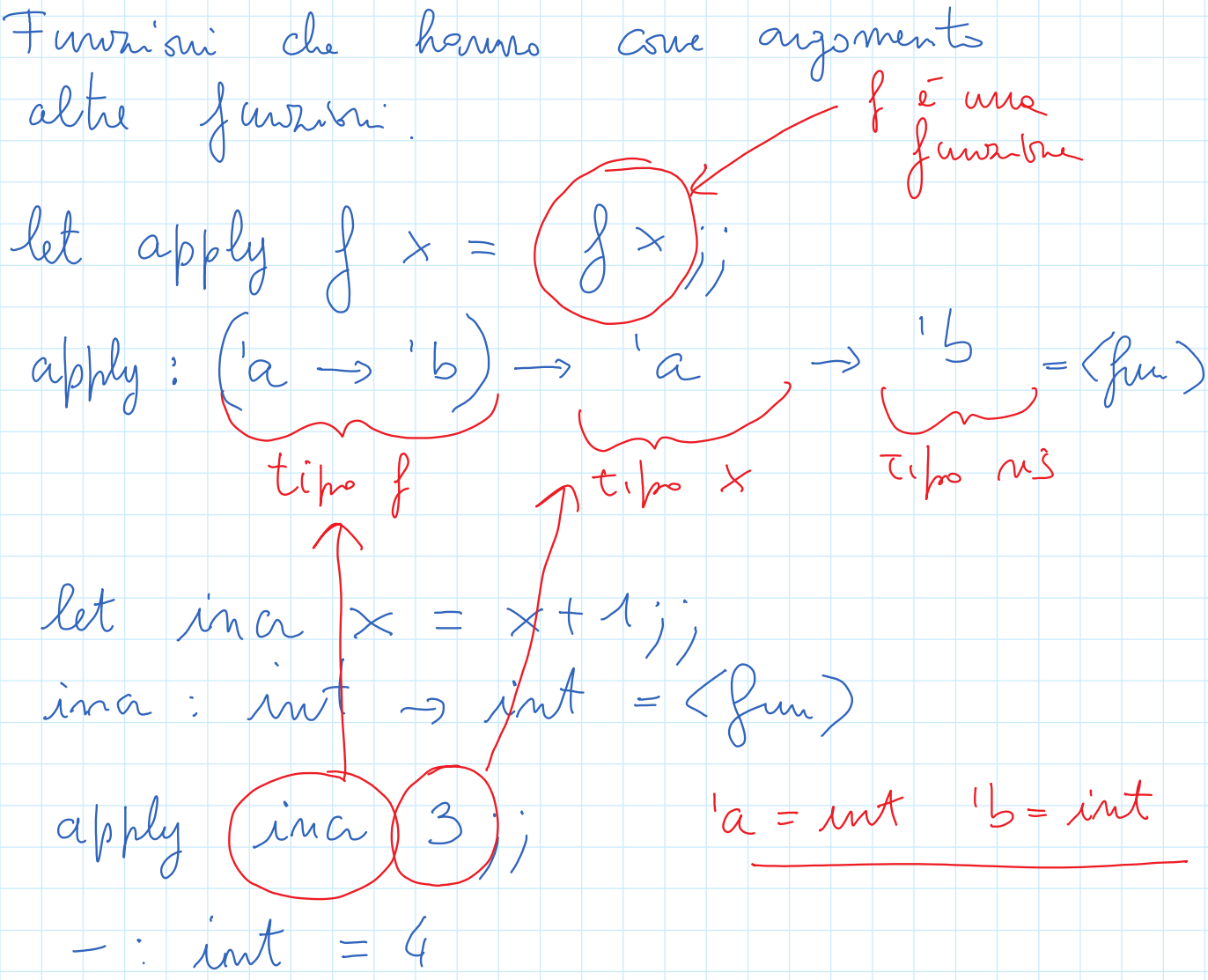
Funzioni di ORDINE SUPERIORE (al primo)

Funzioni che hanno come argomento oppure restituiscono come risultato altre funzioni.

Curryed

```
let somma x y = x+y;;
somma : int -> int -> int = <fun>

let g = somma 4;;
g : int -> int = <fun>

g 5;;
- : int = 9
```

Funzioni che hanno come argomento altre funzioni.



apply : ('a -> 'b) -> 'a -> 'b

let sum (x, y) = x + y ;;

sum : int * int -> int = <fun>

apply sum (3,4) ;;

'a = int * int

'b = int

- : int = 7

apply : (int * int -> int) -> int * int -> int

sum (3,4)

apply sum (3, true) ;;

errore di tipo

```
let apply (f,x) = f x ;;
apply : ('a -> 'b) * 'a -> 'b = <fun>
```

Altre funzioni di ordine superiore di uso comune.

Predicato

è una funzione che ha come risultato un valore di verità (bool)

```
let pari x = x mod 2 = 0 ;;
pari : int -> bool = <fun>
```

```
let s (x,y) = x+y = 10 ;;
s : int * int -> bool = <fun>
```

Come controllare che un predicato valga su tutti gli elementi di una lista?

```
forall pari [2;4;10] = true
```

forall pari [2;4;10] = true

forall pari [2;3;10] = false

↳ quantificazione universale su liste

forall s [(3,7);(4,6);(5,5)] = true

" " " ;(5,6); "] = false

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

forall : ('a -> bool) -> 'a list -> bool

tipo p — tipo l — tipo ris

forall pari [2;3;4] ;;

pari : int -> bool

'a = int

corretto per i tipi

forall s [(3,7); (4,6)] ;;

s : int * int -> bool

'a = int * int

int * int list

Quantificazione esistenziale su liste

exists pari [3;5;6] = true

exists pari [3;5] = false

exists s [(3,7);(5,9)] = true

" s [(4,7);(5,9)] = false

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

exists : ('a -> bool) -> 'a list -> bool

exists inca [3;4;5];;

int -> int

errore di tipo

map

è una funzione che ha come argomenti una funzione (f) e una lista (l) e dà come risultato la lista l a cui è stata applicata la funzione f su tutti i valori.

es: map inc [3;4;5] = [4;5;6]

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

map : ('a -> 'b) -> 'a list -> 'b list = <fun>

('a -> 'b) : tipo f; 'a list : tipo l; 'b list : tipo ns

map inc [3;4;5]

int -> int

'a = int
'b = int

map pari [3;4;5] ;;

int -> bool

'a = int
'b = bool

- : bool list = [false; true; false]

map

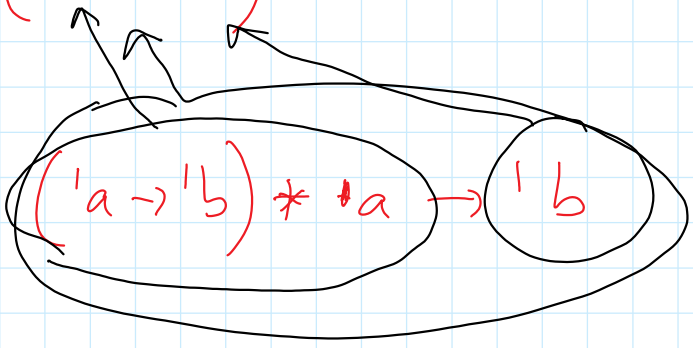
le funzioni possono essere passate come argomenti!

```
let apply (f, x) = f x ;;
apply : ('a -> 'b) * 'a -> 'b = <fun>

let incr1 x = x + 1 ;;
incr1 : int -> int = <fun>

let incr2 x = x + 2 ;;
incr2 : int -> int = <fun>

map apply [(incr1, 5) ; (incr2, 7)] ;;
```

lista di coppie
int -> int * int

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

apply : ('a -> 'b) * 'a -> 'b



map

(map)

$'a = (('a \to 'b) * 'a)$

$'b = 'b$

map apply ( )

il tipo di map diventa:

$((('a \to 'b) * 'a \to 'b) \to (('a \to 'b) * 'a)\ list) \to 'b\ list$

map apply [(inc1, 5); (inc2, 7)]

il tipo di map diventa

$(((int \to int) * int \to int) \to ((int \to int) * int\ list) \to int\ list$

map apply [(inc1, 5); (inc2, 7)];;

- : int list = [6; 9]

filter p l

lascia nel risultato solamente gli elementi di l in cui è vero il predicato p

filter pari [3;4;5;8] = [4;8]

filter s [(3,7);(4,5);(5,5)] =

[(3,7);(5,5)]

```
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

tipo p — tipo l — tipo xs

lunedì 3 dicembre 2018 10:30

foldr f a l

foldr f a $[x_1; x_2; \ldots x_n] =$

$f\ x_1 \ldots \ldots \ldots (f\ x_{n-1}\ (f\ x_n\ a)) \ldots)$

elementi di liste

risultato delle f sugli elementi di liste che seguono

Es

```
let f x y = 1+y ;;
f : int -> int -> int

foldr f 0 [3;4;5] ;;
```

$= f\ 3\ (f\ 4\ (\underbrace{f\ 5\ 0}_{1}))$

$= f\ 3\ (f\ 4\ 1)$

$= f\ 3\ \left(\frac{f\ 4\ 1}{2}\right)$

$= f\ 3\ 2$

$= 3$ lunghezza della lista $[3;4;5]$

nella folds, f ha come argomenti:

- un valore di lista
- il risultato sugli elementi che seguono

$f\ x\ y = 1 + y$

risultato che comprende x

risultato sugli elementi che seguono x

```
f x y = x + y ;;
```

↑

```
foldr f ∅ [3;4;5]
= f 3 (f 4 (f 5 ∅))
= f 3 (f 4 5)
= f 3 9
= 12
```

```
f x y = if pari x then x :: y
                  else y ;;
```

```
foldr f [] [3;4;5]
```

$$= f\ 3\ \Big(f\ 4\ \big(\underbrace{f\ 5\ []}_{[5]}\big)\Big)$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{[4]}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{[4]}$$

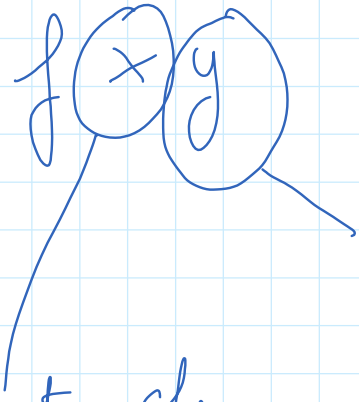
foldr f a l

f ha due argomenti:
- il primo è l'elemento di lista che stiamo esaminando
- il secondo è il risultato dell'operazione che stiamo definendo su tutti gli elementi che seguono!

a è il risultato della operazione che stiamo definendo sulle liste vuote

Vogliamo solamente gli elementi pari

quando la lista è vuota il risultato deve essere [] ⟶ a = []



risultato su tutti gli elementi che seguono x, cioè è la lista di tutti i valori pari che seguono x

elemento che stiamo valutando

$[(3,5);(6,7);(2,3)]$

Vogliamo la lista delle somme di ciascuna coppia.

↪ $[8;13;5]$

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

---

folder f a l = lista di coppie di interi

a = []

```
let f x y = match x with
    (n,m) -> ...
```

let f x y = let (n, m) = x in ....

let f (n, m) y = (n + m) :: y

foldr f [] [(3,5); (5,5); (6,7)] =

f (3,5) (f (5,5) (f (6,7) []))

[13]

[10; 13]

[f; 10; 13]

```
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 ris