```
let rec take n l = match (n, l) with
    (0, l)            -> []                 |
    (n, []) when n>0  -> []                 |
    (n, x::xs) when n>0 -> x::(take (n-1) xs)

let rec drop n l = match (n, l) with
    (0, l)            -> l                  |
    (n, []) when n>0  -> []                 |
    (n, x::xs) when n>0 -> drop (n-1) xs ;;
```

$\forall n \in \mathbb{N}, l \in \text{'a list}.\ (\text{take } n\ l) @ (\text{drop } n\ l) = l$

$\mathbb{N} * \text{'a list}$: dominio

$$(n, l) \sqsubset (n', l') \equiv$$

$$(n' \neq 0 \wedge n = n' - 1) \wedge$$
$$(l' \neq [\,] \wedge l = tl\ l')$$

CASO BASE 1

$(\text{take } \emptyset\ l)\ @\ (\text{drop } \emptyset\ l)$

$=$ { def. di take e drop, 1° patt }

$[\,]\ @\ l$

$=$ { def di @ }

l

CASO BASE 2

$(\text{take } n\ [\,])\ @\ (\text{drop } n\ [\,])$

$=$ { ipotesi $n \in \mathbb{N}$, $n > 0$, 2° patt. }

$[\,]\ @\ [\,]$

$=$ { def. di @ }

$[\,]$

CASO INDUTTIVO

Ip: $n>0$

$$\underbrace{(\text{take } (n-1)\ xs)\ @\ (\text{drop } (n-1)\ xs) = xs}_{\text{IP. INDUTTIVA}}$$

$$\Longrightarrow (\text{take } n\ x::xs)\ @\ (\text{drop } n\ x::xs) = x::xs$$

$$(\text{take } n\ x::xs)\ @\ (\text{drop } n\ x::xs)$$

$= \{$ Ip. $n>0$, 3° pattern di take e drop $\}$

$$(x::(\text{take } (n-1)\ xs))\ @\ (\text{drop } (n-1)\ xs)$$

$= \{$ proprietà $(x::xs)\ @\ ys = x::(xs\ @\ ys)$ $\}$

$$x::(\underbrace{(\text{take } (n-1)\ xs)\ @\ (\text{drop } (n-1)\ xs)})$$

$= \{$ Ip. induttiva $(n-1, xs) < (n, xs)$ $\}$

$$x::xs$$

REVERSE (rovesciare) di una lista

1) $\text{rev } [\,] = [\,]$

2) $$\text{rev } [x_1; x_2; \cdots; x_n] = \underbrace{\text{rev } [x_2; \cdots; x_n]}_{[x_n;\, x_{n-1};\, \cdots;\, x_2]} @ [x_1]$$

$$[x_n; x_{n-1}; \cdots; x_2] @ [x_1]$$

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

rev : 'a list → 'a list

$$\text{rev } [1;2;3]$$

$$= \quad \{2° \text{ pattern di rev}\}$$

$$\underline{(\text{rev } [2;3])} @ [1]$$

$$= \quad \{2° \text{ pattern di rev}\}$$

$$((\text{rev } [3]) @ [2]) @ [1]$$

$$= \quad \{2° \text{ pattern di rev}\}$$

$$(((\text{rev } [\,]) @ [3]) @ [2]) @ [1]$$

$$= \quad \{1° \text{ pattern di rev}\}$$

$$([\,] @ [3]) @ [2] @ [1]$$

$$= \quad \{\text{calcoli}\}$$

$$[3;2;1]$$

REVERSE CON ACCUMULATORE

reverse l l1

accumulo passo passo il risultato che voglio ottenere

quando l si svuota, il secondo argomento rappresenta il risultato del calcolo

reverse [1;2;3] [ ]

reverse [2;3] 1 :: [ ]

reverse [3] 2 :: 1 :: [ ]

reverse [ ] 3 :: 2 :: 1 :: [ ]

```
let rec reverse l l1 = match l with
      [] -> l1
    | x::xs -> reverse xs x::l1 ;;
```

reverse : 'a list -> 'a list -> 'a list

Relazione di precedenza indotta

$$(xs,\ x::l1) \sqsubset (x::xs,\ l1)$$

$$\boxed{\forall l, l'.\ (\text{reverse } l\ l') = (\text{rev } l) @ l'}$$

```
# let rev1 l = reverse l [] ;;
rev1 : 'a list -> 'a list
```

FUNZIONI DI ORDINE SUPERIORE

Quantificazione "universale" $\forall x \in l.\ p(x)$

```
let rec forall p l = match l with
      [] -> true
   | x :: xs -> (p x) & (forall p xs) ;;
```

forall : ('a → bool) → 'a list → bool

p è un predicato, cioè una funzione che restituisce un booleano

```
# let magØ x = x > 0 ;;
  magØ : int -> bool = <fun>

# forall magØ [1;3] ;;
  - : bool = true

# forall magØ [-1; 1; ......; 1000000] ;;
  - : bool = false
```

Calcolo (magØ -1) & (magØ 1) & . . . .
. . . . & (magØ 1000000) & true
= false

forall magØ [1; 3]
= {2° patt.}
(magØ 1) & (forall p [3])
= {2° patt.}
(magØ 1) & (magØ 3) & forall magØ []
= {1° patt.}
(magØ 1) & (magØ 3) & true
= {def. di magØ}
true

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

QUANTIFICATORE ESISTENZIALE $\exists x \in l \,.\, p(x)$

```
let rec exists p l = match l with
        []  ->  false
  | x :: xs  ->  (p x) or (exists p xs);;
```

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

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

MAP : applicare una stessa operazione a tutti gli elementi di una sequenza

$$[x_1; \dots ; x_n] \xrightarrow{f} [f\,x_1; \dots\dots ; f\,x_n]$$

(f, x) [

(f, x :: xs)

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

$$\text{map} : \underbrace{('a \to 'b)}_{f} \to \underbrace{'a\ list}_{l} \to \underbrace{'b\ list}_{\text{risu.}}$$

Funzione che azzera tutti gli elementi negativi di una lista di interi

$$\text{azzeraneg } [-1; 2; 3; -5; 4] = [\emptyset; 2; 3; \emptyset; 4]$$

```
let rec azzeraneg l =  match l with
        []                    ->        []
  | x :: xs   when  x >= Ø  ->      x :: (azzeraneg xs)
  | x :: xs   when  x < Ø   ->      Ø :: (azzeraneg xs)
```

$\circ$ ———————— $\circ$

azzeraneg : int list $\longrightarrow$ int list

\# let g x = if x < 0 then 0 else x ;;

g : int → int

\# let otteraneg l = $\overbrace{\text{map g l}}$;;

otteraneg : $\underset{\text{int}}{\cancel{'a}}$ list → $\underset{\text{int}}{\cancel{'b}}$ list

map : ('a → 'b) → $\underbrace{\text{'a list}}$ → $\underbrace{\text{'b list}}$

$\quad\quad$ ‖ $\quad\quad$ ‖

g : int → int

DEFINIZIONI LOCALI let ... in

```
# let azzeraneg l = let g x = if x<0 then 0 else x
                    in
                       map g l ;;

# g 3 ;;
  ^
   unbound

# azzeraneg [-1 ; 2 ; -3] ;;
  - : int list = [0 ; 2 ; 0]
```

let E in E'

i nomi definiti in E sono VISIBILI in E' ma non altrove

```
# let y = let x = 10 in x + 1 ;;
  y : int = 11

# x + 2 ;;
  ^
  unbound

# y + 2 ;;
  - : int = 13
```

FILTER : cancella da una sequenza tutti gli elementi che NON soddisfano una data proprietà

```
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 : $('a \to bool) \to 'a\ list \to 'a\ list$

```
# filter mago [-1;2;3;-5];;
- : int list = [2;3]

# let mago x = x > 0 ;;
```