mercoledì 22 novembre 2017 11:17

liste CAML

sequenze omogenee (di valori dello stesso tipo) di qualsiasi lunghezza

[] liste vuote

[3; 4; 5]

- : int list = [3; 4; 5]

Operatore (cons) in CAML ::

che ha come operandi un elemento di lista e una lista e costruisce la lista con l'aggiunta di un elemento in testa

es: 3 :: [3; 4] = [3; 3; 4]

```
3::[];;
- : int list = [3]

3::(3::[]);;
- : int list = [3; 3]
```

= associa a destra

Si possono avere liste di oggetti di qualsiasi tipo!

mercoledì 22 novembre 2017 11:24

```
(3,4) :: [];;
```

coppia di interi

```
- : int * int list = [(3,4)]
```

↑ prodotto cartesiano

A, B insiemi

$$A \times B = \{ (a,b) \mid a \in A \wedge b \in B \}$$

```
(4,5) :: [(3,4)];;
- : int * int list = [(4,5); (3,4)]
```

↑ virgole ↑ punto e virgole

```
(3.5, 4) :: [];;
- : float * int list = [(3.5, 4)]
```

può essere qualsiasi

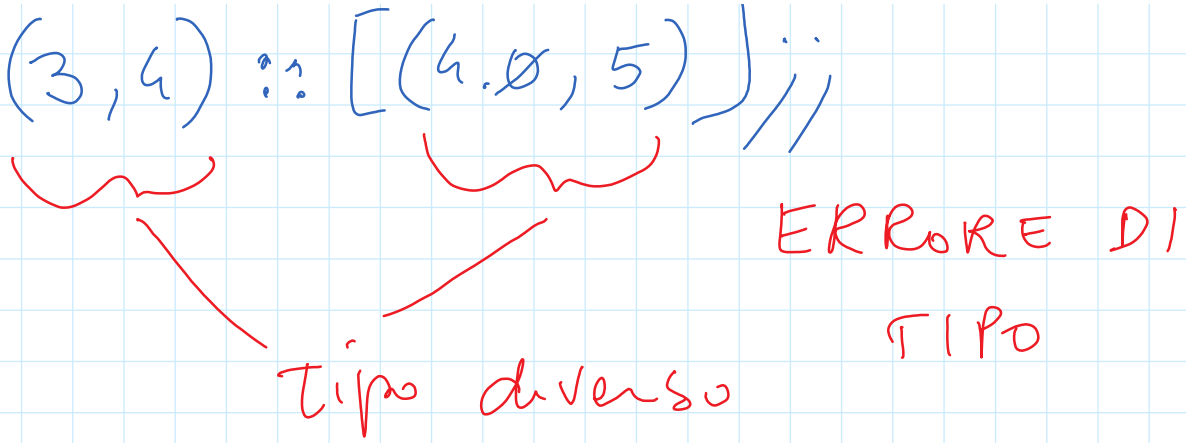
```
[3; 4] :: [];;
```

liste

elementi di liste

```
- : int list list = [[3; 4]]
```

```
let g n = n+1;;
g : int -> int = <fun>

[g];;
- : (int -> int) list = [<fun>]
```


(3,4) :: [(4.0,5)];;
ERRORE DI TIPO
tipo diverso

Su liste sono predefinite alcune funzioni:

```
hd ;;
- : 'a list -> 'a = <fun>
```

head (testa della lista) - primo elemento di una lista

```
hd [3;4];;
- : int = 3

hd [[3;4];[3];[]];;
- : int list = [3;4]

[[];[]];;
- : 'a list list = [[];[]]

hd [];;
Undefined
```

tl — tail (coda della lista) la lista senza il primo elemento.

```
tl [3;4;5];;
- : int list = [4;5]

tl;;
- : 'a list -> 'a list = <fun>
```

Operatore di concatenazione @

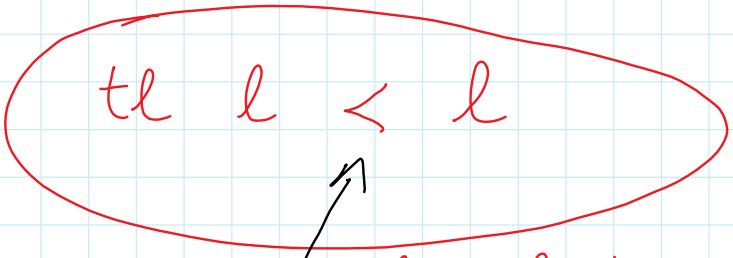
```
[3;4] @ [4;5];;
- : int list = [3;4;4;5]

[3;3] @ [3];;
- : int list = [3;3;3]

tl [3;3;3];;
- : int list = [3;3]
```

Definire una funzione che conta quanti sono gli elementi di una liste.

mercoledì 22 novembre 2017 11:48

len [3;4;3] = 3

Non c'è altro modo che definire len ricorsivamente

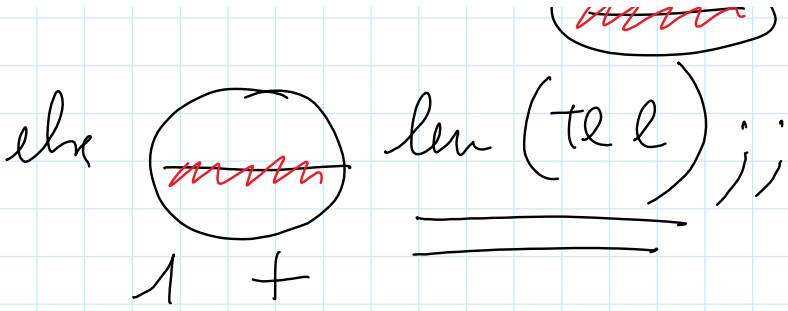


l'insieme delle liste con la precedenza < è ben fondato!!

le definizioni ricorsive hanno quasi sempre la struttura:

- definire la funzione su liste vuote
- supponendo di saper definire la funzione su tl l riesco a definirla su l.

let rec len l = if l=[] then 0




```
let rec len l = if l=[] then 0
                else 1+ len (tl l) ;;
```

Definire una funzione ricorsiva che cancelli l'ultimo elemento di una lista, se c'è. (canc)

canc [3;4;5] = [3;4]
canc [3] = []
canc [] = []

---

let rec canc l =
  if l = [] then []
  else if tl l = [] then []
  else hd l :: canc (tl l)

canc [3;4] = []

so cancellare l'ultimo elemento della lista (tl l)

canc [3;4;5];;
= {def canc, 3° caso}
3 :: canc [4;5];;
= {def canc, 3° caso}
3 :: 4 :: canc [5];;

3 :: 4 :: conc [5] ;;

= { def conc, 2° caso }

3 :: 4 :: []

= { notazione }

[3; 4]

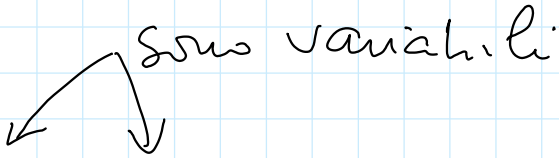
mercoledì 22 novembre 2017 12:22

NOTAZIONE per la definizione di funzioni:

Concetto di PATTERN MATCHING

Cercare di rendere uguale un pattern (modello che contiene delle variabili) a un valore, instanziando opportunamente le variabili.

Esempio



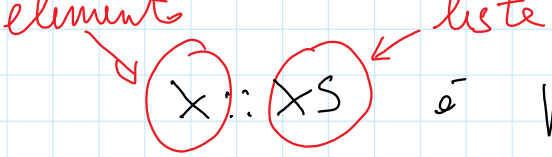
PATTERN $x :: xs$

può essere reso uguale alla lista $3 :: 4 :: 5 :: []$ instanziando x e xs?

Sì! Basta instanziare x a 3 e xs alla lista $4 :: 5 :: []$

$x = 3$ $\quad x :: xs = 3 :: 4 :: 5 :: []$

$xs = 4 :: 5 :: []$ $\quad x :: xs = [3; 4; 5]$



è una lista uguagliando a $[3] = 3 :: []$

$x = 3$ $\quad x :: xs = 3 :: [] = [3]$

xs = [ ]

x :: xs non è uguagliabile a [ ]

pattern x è uguagliabile a qualsiasi valore.

I pattern possono non contenere variabili (servono a poco)

es: [3] è uguagliabile solo al valore [3]

Il pattern [] si uguaglia solamente a [].

Il pattern [x] = x :: []
si uguaglia solamente a liste di un solo elemento

[3;4] ≠ x :: []

[x] = x :: []

non si può uguagliare [3;4]

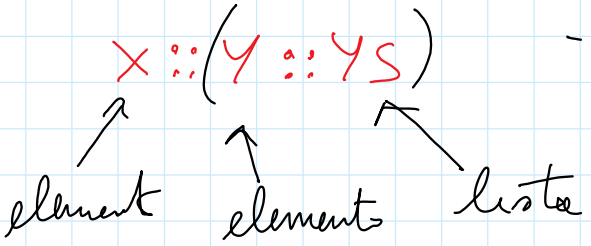
x = [3;4]

quindi il pattern diventa

[3;4] :: [] = ~~[3;4]~~

≠

= [[3;4]]



è uguagliabile a:

[]	no
[3]	no
[3;4]	sì
[3;4;5]	sì
[3;4;5;6]	sì

x=3 y=4 ys=[]

x=3 y=4 ys=[5]

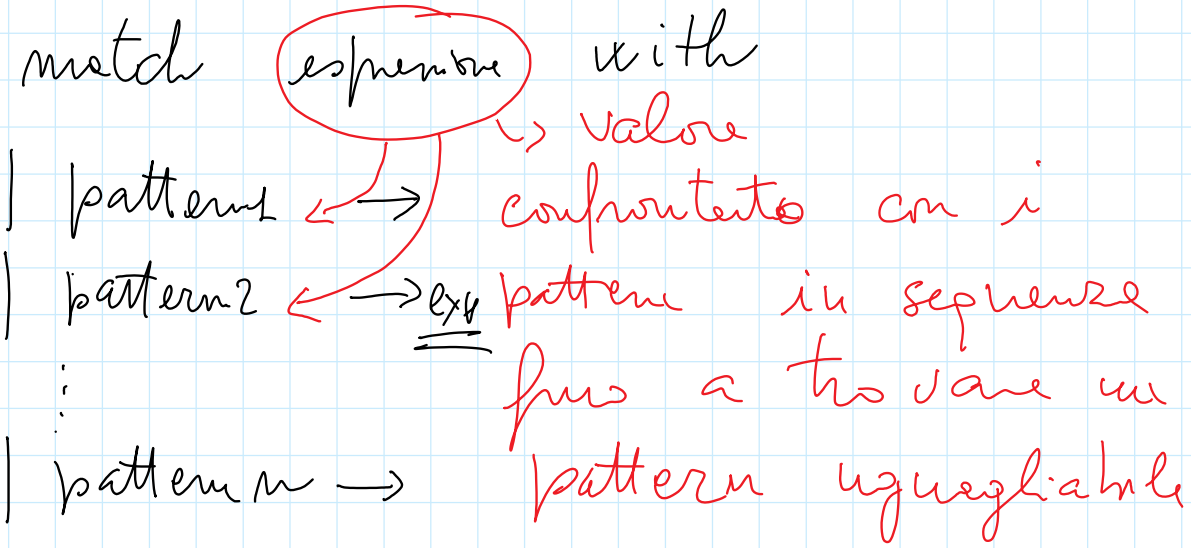
x=3 y=4 ys=[5;6]

x::y::xs è uguagliabile a liste di almeno due elementi!

mercoledì 22 novembre 2017 12:38

Come vi sono i pattern nelle def. di funzioni:

Espressione match



viene eseguita (valutata) l'espressione relativa al primo pattern uguagliabile!

```
match [3;4] with
  [] -> []
| x::xs -> x ;;
- : int = 3
```

x = 3
xs = [4]

->

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

/= hd l

== tl l

```
let rec conc l = match l with
    []          -> []
  | [x]         -> []
  | x::y::ys    -> x :: conc (y::ys) ;;
```

pattern MUTUAMENTE ESCLUSIVI

uguaglia solamente a liste di almeno due elementi

```
    []    ->
  | [x]   ->
  | x::xs ->
```

il risultato sarebbe stato lo stesso.
E' vero che questo pattern si uguaglia anche a liste di un solo elemento.
Le liste di un solo elemento sarebbero state "catturate" da questo pattern.

$[3;4] @ [4] = [3;4;4]$ $\quad [] @ [4] = [4]$

append

Curryed

```
let rec append l1 l2 =
    match l1 with
    [] -> l2
  | x :: xs -> x :: append xs l2 ;;
```

append : 'a list → 'a list → 'a list (finale)

tipo l1 — tipo l2 — tipo xs

append [3;4] [4]

= { def. append, 2° p }

3 :: append [4] [4]

= { def. 2° p }

3 :: 4 :: append [] [4]

= { def. 1° p }

3 :: 4 :: [4]

= { notazione }

[3;4;4]