1 # LISTE IN CAML

La sequenza omogenea di valori di qualunque Tipo

- [] è la lista vuota 'a list
- :: cons aggiunge in Testa

4 :: [] ≡ [4] int list

3 :: 4 :: [] ≡ [3;4] ≡ 3 :: [4]

- [(3, True); (9, false)] int * bool list

[f; g; h] f, g, h ∈ int → bool
(int → bool) list

[[9]; [5; -2]; [4; 7; 8]] int list list

OPERAZIONI SULLE LISTE

- hd (head) restituisce il primo elemento di una lista

```
hd : 'a list → 'a
```

```
# hd [6;5;4];;
- : int = 6
```

- tl (tail) restituisce la lista ottenuta rimuovendo il primo elemento

```
tl : 'a list → 'a list
```

```
# tl [6;5;4];;
- : int list = [5;4]
```

NOTA hd e tl sono definite solo per liste NON VUOTE!

```
# hd [];
```
errore!!

```
# tl [];
```
errore!!

- @ (append) concatenazione di liste

```
# [1;2;3] @ [4;5];;
- : int list = [1;2;3;4;5]
```

FUNZIONE CHE CALCOLA LA LUNGHEZZA DI UNA LISTA

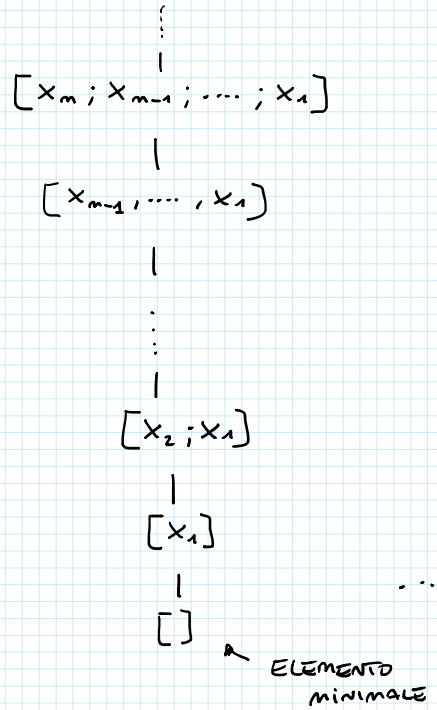
```
# let rec lem l =
      if   l=[]  Then  0
      else  1 + lem (tl l) ;;
```

lem [8;4;5]
1 + lem [4;5]
1 + lem [5]
1 + lem []
||
0
1+1+1+0 = 3

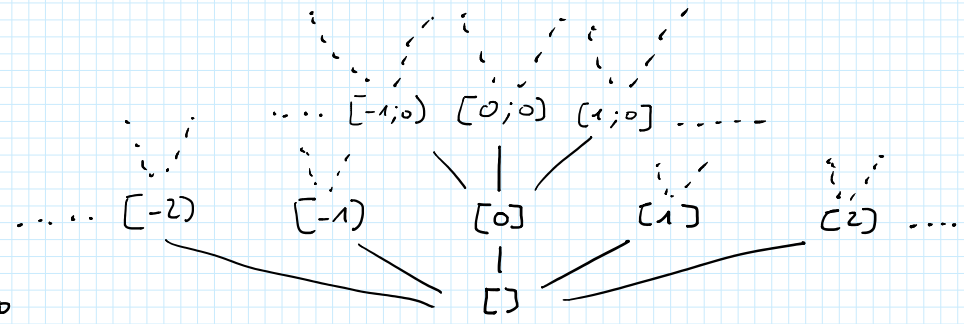
```
lem: 'a list → int = <fun>

# lem [1;5;8];;
- : int = 3
```

vediamo se la relazione di precedenza indotta $\prec_{lem}$ è ben fondata

$$l \prec_{lem} l' \equiv l = tl\ l' \quad (\text{oppure } l' = x :: l)$$



ad esempio su int list



ben fondato!!

giovedì 1 dicembre 2016 08:55

PATTERN MATCHING

costrutto che consente di confrontare il risultato di una espressione con una sequenza di casi possibili:

```
match espressione with
    pattern1 -> espressione1
  | pattern2 -> espressione2
      ⋮
  | patternN -> espressioneN
```

esempio: negazione di un booleano

```
# let negazione e =
    match e with
      True -> false
    | false -> True ;;

negazione : bool -> bool -> <fun>

# negazione (8<2) ;;
- : bool = True
```

Un pattern (modello) può essere:

- una costante: True, false, 0, 1, -1, -5, ... 'a', 'b', ...
- una variabile: x, y, z, ...
- una coppia (o ennupla) di pattern: (x,y) (x,3) (x,y,z)
- una lista vuota: []
- una lista (di pattern) ottenuta usando :: (cons): x :: [] x :: y 3 :: x (x,y) :: z

Un pattern "fa match" con un valore se esiste un modo di istanziare le variabili nel pattern che mi consente di ottenere il valore

PATTERN	VALORI		
3	3	✓	FA MATCH
3	4	✗	NON FA MATCH
x	3	✓	istanzio x=3
(x,y)	(8,4)	✓	x=8 y=4
(x,y)	(8,4,2)	✗	
(x,8)	(2,8)	✓	x=2
(x,8)	(2,9)	✗	
[]	[]	✓	
x :: []	[8]	✓	x=8
8 :: []	[8]	✓	
8 :: []	[9]	✗	
x :: []	[8;9]	✗	
x :: y :: []	[8;9]	✓	x=8 y=9
x :: xs (x: elemento, xs: lista)	[1;2;3]	✓	x=1 xs=[2;3]
	[1]	✓	x=1 xs=[]
	[]	✗	

```
match e with
   p1 → e1
 | p2 → e2
   ⋮
 | pN → eN
```

e viene valutata e il suo risultato viene confrontato con p1, ..., pN IN SEQUENZA

appena si trova in pattern pi che "fa match" con e, il risultato calcolato sarà dato dall'espressione ei corrispondente

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

'a list → inferito guardando i pattern

NOTA:
le variabili usate nel pattern possono essere usate come variabili locali nell'espressione corrispondente.
Il loro valore sarà quello calcolato per verificare che il pattern fa match!

ESEMPIO : funzione che restituisce l'ultimo elemento di una lista

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

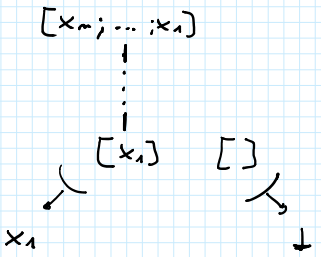
[3;2] → [2) → 2

last : 'a list → 'a = <fun>

```
# last [3;4;5];;
- : int = 5
```

QUESTA FUNZIONE NON È DEFINITA SU [] (PATTERN MATCHING IN QUESTO CASO NON È ESAUSTIVO)

$l \prec_{last} l' \equiv l = tl\ l'$ e $len\ l' \geq 2$



LA RELAZIONE $\prec_{last}$ È BEN FONDATA MA LA FUNZIONE NON È DEFINITA SULL'ELEMENTO MINIMALE

→ cosa succede si invertono i casi nel pattern matching?

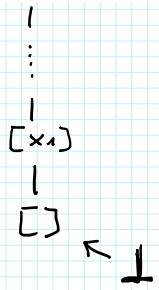
```
# let rec last l
      match l with
        x::xs -> last xs
      | x::[] -> x ;;
```

→ INUTILIZZABILE

```
# last [3;2);;
  errore!
```

$l \prec_{last} l' \equiv l = tl\ l' \quad l' \geq 1$

[xn, ... , x1]



raggiungo sempre il caso non definito !!

ESEMPIO funzione che "rovescia" una lista

[1;2;3] => [3;2;1]

```
# let rec reverse l =
    match l with
      [] -> []
    | x::xs -> (reverse xs)::x
```

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

PROBLEMA! CONS SI ASPETTA UN ELEMENTO E UNA LISTA, NON IL CONTRARIO

NON SI PUÒ FARE

SOLUZIONE: Funzione con accumulatore!

```
# let rec reverse_acc l1 l2 =
    match l1 with
      [] -> l2
    | x::xs -> reverse_acc xs (x::l2);;
```

l1	l2
[1;2;3]	[]
[2;3]	[1]
[3]	[2;1]
[]	[3;2;1]

risultato

IN GENERALE x::l ≡ [x]@l

```
# let reverse l = reverse_acc l [];;
```

reverse