Titolo nota 22/10/2015

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

f

x

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

tipo x — tipo risultato

```
# let h = apply g ;;
  h : int -> int = <fun>

# h 5 ;;
  - : int = 6
```

```
# let s n = n = 3 ;;
  s : int -> bool = <fun>
```

n — ris.

```
# let pippo = apply s ;;
  pippo : int -> bool = <fun>

# pippo 5 ;;
  - : bool = false

# pippo 3 ;;
  - : bool = true
```

ESPRESSIONE CONDIZIONALE

Titolo nota 22/10/2015

```
if e then e1 else e2
```

e: espressione di tipo bool

e1, e2: devono avere lo stesso tipo, che è anche il tipo dell'intera espressione

```
# let m x y = if x > y then x else y;;
m : 'a -> 'a -> 'a = <fun>
```

(bool; stesso tipo)

```
# m 3 4;;
- : int = 4
```

(int)

```
# m 3 'a';;
errore
```

CONDITIONAL AND

Titolo nota 22/10/2015

a cand b = il valore di a se a è falso, altrimenti il valore di b

```
# let cand a b = if (not a) then false else b ;;
  cand : bool -> bool -> bool = <fun>
         (a)    (b)    (risultato)
```

```
# let cong (x,y) = if (x = false or y = false) then false else true ;;
  cong : bool * bool -> bool = <fun>
         (x)   (y)     (ris)
```

```
# let imply x y = if (not x) then true else y ;;
  imply : bool -> bool -> bool = <fun>
          (x)    (y)     (ris)
```

DICHIARAZIONI LOCALI

Titolo nota 08/10/2015

Sia f la funzione che, dato x, restituisce (g x) + 1,
dove g è la funzione che raddoppia il suo argomento.

```
# let f x =
    let g z = 2*z
    in (g x) + 1;;
```

g : int → int (z : int, ris : int)

f : int → int = <fun>
(x : int, ris : int)

```
# f 3;;
- : int = 7
# g 3;;
```

g undefined

```
# let undici =
    let n = 10
    in n+1;;
undici : int = 11
```

sia undici il valore della espressione n+1, dove n è il valore 10

DEFINIZIONI RICORSIVE in CAML.

Titolo nota 08/10/2015

$$fact(n) = \begin{cases} 1 & \text{se } n=0 \\ n * fact(n-1) & \text{se } n>0 \end{cases}$$

$fact : \mathbb{N} \to \mathbb{N}$

```
# let rec fact n = if n=0 then 1 else n * fact (n-1) ;;
        fact : int -> int = <fun>

# fact 3 ;;
  - : int = 6
# fact (-1) ;;
```

errore a tempo di esecuzione !!

RAGIONARE SU FUNZIONI DEFINITE RICORSIVAMENTE

Titolo nota 08/10/2015

```
let rec f n = if n = 0 then 0 else 3 + f (n-1) ;;
```

$f : \mathbb{N} \to \mathbb{N}$

$\forall n \in \mathbb{N}.\ f\ n = 3 * n$

① facciamo vedere che sul dominio di interesse $\mathbb{N}$, la relazione di precedenza indotta dalla funzione è ben fondata

② dimostriamo per induzione ben fondata la proprietà

$x \sqsubset_f y \equiv x = y - 1$ è ben fondata su $\mathbb{N}$

Caso base

$f(0)$

$= \{\text{def. } f\}$

0

$= 3 * 0$

Caso induttivo: sia $x > 0$, $x \in \mathbb{N}$, $x \neq 0$

$f(x)$

$= \{\text{def. di } f, \text{ramo else}\}$

$3 + f(x-1)$

$= \{\text{Ip. induttiva: } f(x-1) = 3*(x-1), \text{ perché } x-1 \sqsubset_f x\}$

$3 + 3*(x-1) = 3 + 3*x - 3 = 3*x$

```
⋮
4
3
2
1
0
```

ALTRO ESEMPIO

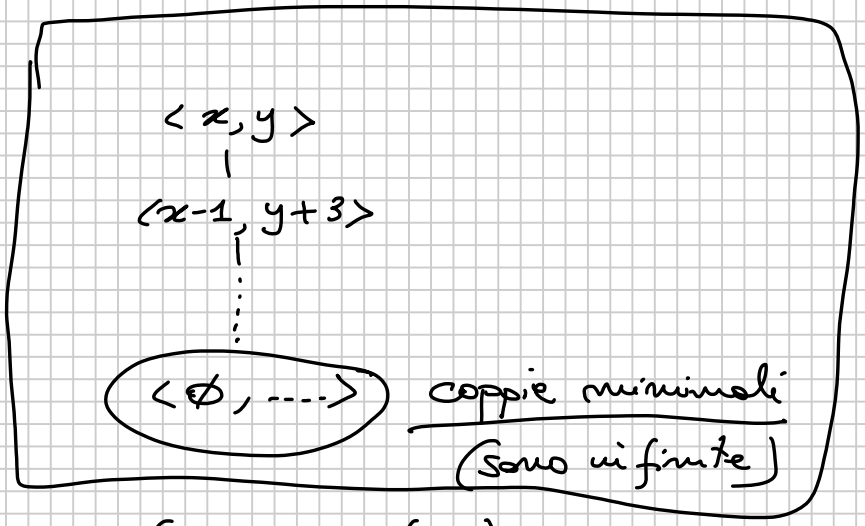
Titolo nota 08/10/2015

```
let rec f n m = if n = 0 then m
                else f (n-1) (m+3);;
```

$f : \underbrace{int}_{n} \to \underbrace{int}_{m} \to \underbrace{int}_{ris} = \langle fun \rangle$

$\forall n, m \in \mathbb{N}.\ f\ n\ m = m + 3 * n$

$\langle x, y \rangle \sqsubset_f \langle z, w \rangle \equiv x = z - 1 \wedge y = w + 3$
$(z = x + 1 \wedge w = y - 3)$



Caso base

```
  f 0 m
= {rami then}
  m
= {calcolo}
  m + 3*0
```

Caso induttivo : $\langle n, m \rangle$ con $n \neq 0$

```
  f n m
= {rami else}
  f (n-1) (m+3)
= {⟨n-1, m+3⟩ ⊏ ⟨n, m⟩
   Ip. ind: f (n-1) (m+3) = (m+3) + 3*(n-1)}
= (m+3) + 3*(n-1)
= {calcolo}
  m + 3 + 3*n - 3
```

Titolo nota 08/10/2015

Poiché $\forall n,m.\ f\ n\ m = m + 3*n$

l'applicazione

$$f\ n\ \emptyset = \emptyset + 3*n = 3*n$$

Il calcolo di $f\ n\ m$, quando viene iniziato con $m = \emptyset$, accumula nel secondo argomento il prodotto $3*n$

È il primo esempio di funzione con un argomento che "accumula" il calcolo desiderato.

```
# let g n =
    let rec f n m = if n=0 then m else f (n-1) (m+3)
    in
      f n 0 ;;
g : int -> int = <fun>
```

Abbiamo appena dimostrato

$$\forall n \in \mathbb{N}.\ g\ n = 3*n$$

LISTE in CAML

Titolo nota 08/10/2015

La liste in CAML è un tipo "predefinito" attraverso una costante predefinita

`[]` la lista vuota

e attraverso un costruttore di tipo (cons) la cui sintassi CAML è `::`

`::` : 'a * 'a list → 'a list

Il costruttore `::` restituisce una NUOVA LISTA in cui la testa è il primo argomento e la coda è il secondo argomento.

'a list è il tipo delle liste di oggetti omogenei di tipo 'a

Le liste in CAML corrispondono alle pile che abbiamo usato nella semantica di C

$$\Pi = \{\Omega\} \cup \{ f.\pi \mid f \text{ è un frame, e } \pi \in \Pi \}$$

Ω ↓ `[]` $f.\pi$ ↓ `::`

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

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

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

# 3 ;;
- : int = 3
```

In generale, una lista ottenuta in questo modo

$x_1 :: x_2 :: x_3 :: \ldots :: x_n :: []$

si può scrivere più sinteticamente così

$[x_1; x_2; \ldots; x_n]$

```
4 :: 3 :: []

# ['a'; 'b'] ;;
- : char list = ['a'; 'b']

# ['a'; 3]
```

← ERRORE gli elementi di una lista devono avere lo stesso tipo.

Titolo nota 08/10/2015

```
# let f x = x+1;
  f: int -> int = <fun>
# let g x = x+2;
  g : int -> int = <fun> .

# f :: [];;
  -:(int -> int) list = [<fun>]

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

```
:: : 'a * 'a list -> 'a list
f : int -> int
[] : 'a list
f :: [] : (int -> int) list
```

ESEMPI

Titolo nota 08/10/2015

```
# [(10, 'a'); (-1; '3')];;
```

int char int char

(int * char) (int * char)

```
-: (int * char) list = [ . - - - ]
```

```
# [3+2; 8-5; ∅];;
```

int int int

```
-: int list = [5; 3; ∅]
```

```
# [[1]];;
```

int list

```
-: (int list) list = [[1]]
```

```
# [[]; [1]; [-2;3]];;
```

int list int list int list

```
# [['a']; [1]];;
```

char list int list

NO