lunedì 28 novembre 2016 09:44

2

Riepilogo CAML

→ Tipi int, float, bool, char, string

'a'

→ let per dichiarare nomi
- variabile let x = 5;;
- funzioni let f x = x + 1;;

→ funzioni con più parametri
- let f (x,y) = x+y;;
- let f x y = x+y;;

let g = f 0

consente l'applicazione parziale

→ funzioni polimorfe let i x = x;;

i : 'a → 'a = <fun>

es. # let maggiore x y = x > y;;

maggiore : 'a → 'a → bool = <fun>

'a: Tipo x; 'a: Tipo y; bool: Tipo ritornato

es. # fst;;

let fst (x,y) = x;;

- : 'a * 'b → 'a = <fun>

martedì 29 novembre 2016 08:54

FUNZIONI DEFINITE PER CASI

$$abs(x) = \begin{cases} x & \text{se } x \geq 0 \\ -x & \text{se } x < 0 \end{cases}$$

→ uso una ESPRESSIONE CONDIZIONALE

ESPRESSIONI DELLO STESSO TIPO

```
if x>=0 Then x else -x
```

ci vogliono sempre!

```
# let abs x = if x>=0 Then x else -x ;;
abs : int -> int = <fun>

# abs (-3);;
```

ricordarsi le parentesi!!

ALTRO ESEMPIO

$$max(x,y) = \begin{cases} x & \text{se } x>y \\ y & \text{altrimenti} \end{cases}$$

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

('a x, 'a y, 'a risultato)

DICHIARAZIONI LOCALI

$$areacerchio(r) = r^2 \cdot \pi \qquad \text{dove } \pi = 3,14$$

```
# let areacerchio r =
    let pi = 3.14 in
      r *. r *. pi ;;
```

|| limito la visibilità di pi al corpo di areacerchio! non potrò usare pi nel seguito...

$f(x) = g(x) + 1$ dove $g(z) = 2 * z$

funzione ausiliaria

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

→ g: int (z) → int (ris)

parentesi superflue (applicazione ha precedenza su +)

f: int (x) → int (ris) = <fun>

```
# f 3;;
```
✓

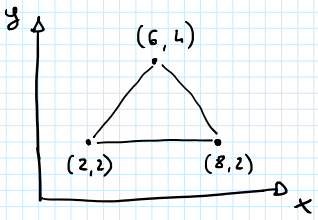
```
# f (-3);;
```
✓

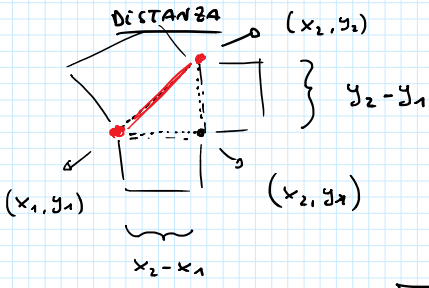
```
# g 2;;
```
✗

g unbound (non definita)

ESEMPIO PIU' COMPLESSO

→ Scriviamo una funzione che calcola il perimetro di un Triangolo date le coordinate dei Tre angoli





$$dist\left((x_1,y_1),(x_2,y_2)\right) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

```
# let perimetro p1 p2 p3 =
    let distanza (x1,y1) (x2,y2) =
        sqrt ( (x2-.x1) *. (x2-.x1) +. (y2-.y1) *. (y2-.y1) )
    in distanza p1 p2 +.
       distanza p2 p3 +.
       distanza p3 p1 ;;
```

FUNZIONE PREDEFINITA

perimetro : float * float → float * float → float * float → float = <fun>

```
# perimetro (0.0, 0.0) (3.0, 0.0) (0.0, 4.0) ;;
- : float = 12.0
```

DEFINIZIONI RICORSIVE

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

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

```
# let rec fact n = if n=0 then 1
                   else n * fact (n-1) ;;
```

REC DICE CHE LA FUNZIONE È RICORSIVA

CI VOGLIONO

fact n - 1
=
(fact n) - 1 ✗

su numeri negativi il programma non Termina

$(\mathbb{Z}, \prec_{fact})$ non ben fondato!!

$x \prec_{fact} y \equiv x = y-1$

5 4 3 2 1 0 =

-3 -4 -5 ... ↓

per renderla Totale su intT dovrei aggiungere un caso per $n<0$

martedì 29 novembre 2016 08:54

ALTRO ESEMPIO
- PER RAGIONARE SU FUNZIONI RICORSIVE
- FUNZIONE CHE USA UN ACCUMULATORE

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

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

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

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

$\langle n, m \rangle \quad n > 0$
|
$\langle n-1, m+3 \rangle$
|
⋮
$\langle 0, k \rangle$
↓
$k \in \mathbb{N}$

INFINITI MINIMALI

OK BEN FONDATA !!

DIMOSTRIAMO LA PROPRIETÀ PER INDUZ. BEN FONDATA

CASO BASE $n = 0$ $m = K$ $K \in \mathbb{N}$

$f\ 0\ k$
$= \{ \text{def. } f \}$
k
$=$
$k + 3 \cdot 0$ ✓

CASO INDUTTIVO $n > 0$ $m = k$

$f\ n\ m$
$= \{ \text{def. } f \text{ nano else} \}$
$f\ (n-1)\ (m+3)$
$= \{ \text{ip. induttiva} \}$
$(m+3) + 3 \cdot (n-1)$
$=$
$m + 3 + 3n - 3$
$=$
$m + 3n$ ✓

nella funzione precedente m ha svolto il ruolo di "accumulatore" del risultato mentre n ha determinato la successione delle chiamate ricorsive

- Poiché $\forall n, m \in \mathbb{N}$ f n m = m + 3n

  quindi f n 0 = 0 + 3 n = 3 n

Se siamo interessati a calcolare 3 n possiamo usare f come funzione ausiliaria

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

→ la funzione ricorsiva accumula il risultato

LISTE IN CAML

La lista in CAML è un Tipo di dato predefinito !!

[] rappresenta la lista vuota

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

'a = variabile di Tipo; list = Tipo lista

'a list = lista di elementi di qualunque Tipo

:: detto cons consente di costruire una lista aggiungendo un elemento in Testa ad una lista esistente

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

[4;3] ≡ 4::[3] ≡ 4::3::[]

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

# [True ; false ; True];;
- : bool list = [.....]

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

f e g hanno lo stesso Tipo int→int

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

# [3+2 ; 6-4];;
- : int list = [5, 2]

# [[3;2] ; [4;1;2]] ;;
```

- : int list list = [...]