$$\Big(P(\emptyset) \wedge \big(\forall n \in \mathbb{N} \,.\, P(n) \Rightarrow P(n+1)\big)\Big)$$

$$\Longrightarrow$$

$$\big(\forall n \in \mathbb{N} \,.\, P(n)\big)$$

Principio di induzione sui naturali

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

$$f(n) = \begin{cases} 3 & \text{se } n = 0 \\ f(n-1) + 2 & \text{se } n > 0 \end{cases}$$

$$\big(\forall n \in \mathbb{N} \,.\, f(n) = 2 \cdot n + 3\big)$$

Caso base

$$f(\emptyset) = 2 \cdot \emptyset + 3$$

$f(\emptyset)$

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

3

$= \{ \text{calcolo} \}$

$2 \cdot \emptyset + 3$

Caso induttivo

$f(n) = 2 \cdot n + 3 \Rightarrow f(n+1) = 2 \cdot (n+1) + 3$

ip. induttiva

$f(n+1)$

$= \{ \text{def } f,\ n+1 > \emptyset \}$

$f(n) + 2$

$= \{ \text{ip. indutt.} \}$

$2 \cdot n + 3 + 2$

$= \{ \text{calcolo} \}$

$2 \cdot (n+1) + 3$

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

$$f(n,m) = \begin{cases} 3 \cdot n + 1 & \text{se } m = 0 \\ 2 \cdot m + 1 & \text{se } n = 0 \\ f(n-1, m-1) + 5 & \text{se } n > 0 \text{ e } m > 0 \end{cases}$$

$$\left(\forall n, m \in \mathbb{N} \,.\, f(n,m) = 3n + 2m + 1 \right)$$

$$P(n,m)$$

PRINCIPIO DI INDUZIONE BEN FONDATA

Una relazione di precedenza (indicata comunemente come: $\prec$, $\sqsubset$)
è una relazione non riflessiva
$(a \not\prec a)$.

dato un insieme A e una relazione di precedenza $\prec$. $(A, \prec)$
$a, b \in A$ $(a \prec b)$ si dice che

a precede b

$(A, \preceq)$ $(a \preceq b)$ si dice che a precede b

Dato un insieme A e una relazione di precedenza $\prec$

$(A, \prec)$ si dice BEN FONDATO sse non esistono catene infinite discendenti secondo $\prec$

$(\mathbb{Z}, \leq)$ non è ben fondato

$(\mathbb{N}, <)$ ben fondato!

(perché non ci sono catene infinite discendenti)

mercoledì 20 novembre 2019 11:32

INDUZIONE BEN FONDATA

Dato un insieme A e una relazione di precedenza $\prec$. Date una proprietà P su A.
Se $(A, \prec)$ è ben fondato allora:

- Se P è vera su tutti gli elementi minimi secondo $\prec$. (ci possono essere più elementi minimi che si chiamano MINIMALI)
- Se supponendo vera la proprietà su tutti gli elementi che precedono, secondo $\prec$, un elemento qualsiasi $n \in A$, si riesce a dimostrare la proprietà su n

Allora la proprietà P è vera su tutti gli elementi di A.

mercoledì 20 novembre 2019 11:39

11:22

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

relazione di precedenza INDOTTA dalla definizione ricorsiva

$$f(n,m) = \begin{cases} 3 \cdot n + 1 & \text{se } m = 0 \\ 2 \cdot m + 1 & \text{se } n = 0 \\ f(n-1, m-1) + 5 & \text{se } n > 0 \text{ e } m > 0 \end{cases}$$

$$\left(\forall n, m \in \mathbb{N} \,.\, f(n,m) = 3n + 2m + 1 \right)$$

$\mathbb{N} \times \mathbb{N}$

$$\left(\forall n, m, n', m' \in \mathbb{N} \,.\, (n,m) \prec (n',m') \equiv n = n' - 1 \wedge m = m' - 1 \right)$$

$(4,5)$
$\downarrow$
$(3,4)$
$\downarrow$
$(2,3)$
$\vdots$

$(10,20)$
$\downarrow$
$(9,19)$
$\downarrow$
$(8,18)$
$\vdots$

$(2,3) \prec (3,4)$

Quali sono i minimali di $(\mathbb{N} \times \mathbb{N}, \prec)$?

$(\forall n \in \mathbb{N}.\ (0, n))$ $\quad$ $(\forall n \in \mathbb{N}.\ (n, 0))$

minimali

l'argomento della chiamata ricorsiva precede l'argomento della funzione.

Nel nostro caso

$$(n-1, m-1) < (n, m)$$

che formalizzato diventa

$(\forall n, m, n', m' \in \mathbb{N}.$

$(n, m) < (n', m') \equiv$

$n = n' - 1 \ \wedge \ m = m' - 1)$

Caso base 1 $(n, 0)$

$$f(n, 0) = 3 \cdot n + 2 \cdot 0 + 1$$

$f(n, 0)$

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

$3 \cdot n + 1$

$= \{ \text{calcolo} \}$

$3 \cdot n + 2 \cdot 0 + 1$

Caso base 2 $(0, m)$

$$f(0, m) = 3 \cdot 0 + 2 \cdot m + 1$$

$f(0, m)$

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

$2 \cdot m + 1$

$= \{\text{calcolo}\}$

$3 \cdot \emptyset + 2 \cdot m + 1$

Caso induttivo — ip. induttiva

$$f(n,m) = 3 \cdot n + 2 \cdot m + 1 \Longrightarrow$$

$$f(n+1, m+1) = 3 \cdot (n+1) + 2 \cdot (m+1) + 1$$

$f(n+1, m+1)$

$= \{$ def $f,\ n+1, m+1 > \emptyset \}$

$f(n,m) + 5$

$= \{$ ip. induttiva $\}$

$3 \cdot n + 2 \cdot m + 1 + 3 + 2$

$= \{$ calcolo $\}$

$3 \cdot (n+1) + 2 \cdot (m+1) + 1$

Liste CAML

le liste è una sequenza dinamica (posso aggiungere o togliere valori) di elementi dello stesso tipo.

In CAML le liste si indica con:

[-2 ; 3 ; 10 ; -5]

questa è una lista di 4 elementi interi

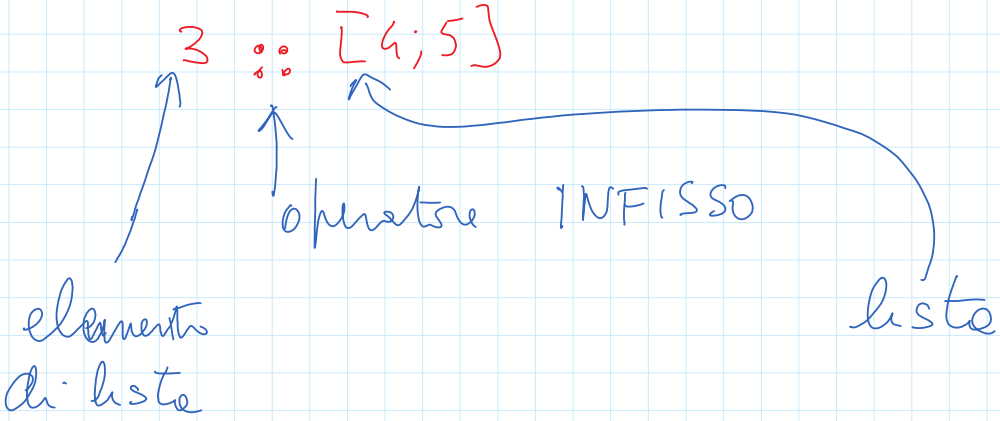
```
# [-2; 3; 10; -5];;
- : int list = [-2; 3; 10; -5]

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

# [3; 3.5];;
```

int ← 3 ; 3.5 → float

errore di tipo

Sulle liste è definito un operatore di base (costruttore di valori)

cons si indica in CAML ::



il risultato di 3 :: [4;5] è la nuova lista [3;4;5]

In CAML le notazioni

3 :: [4;5]

e

[3;4;5]

sono EQUIVALENTI

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

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

# 3 :: (4 :: [5]);;
- : int list = [3;4;5]

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

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

3 :: 4 :: 5 ;;

deve essere una lista di interi

errore di tipo

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

# (3,5) :: (4,6) :: [];;
- : int * int list = [(3,5); (4,6)]

# (3,"ab") :: [];;
- : int * string list = [(3,"ab")]
```

```
# [3;5] :: [] ;;
- : int list list = [[3;5]]

# [3;5] :: [5] :: [] ;;
- : int list list = [[3;5]; [5]]
```

liste di interi

```
# [3;5] :: [5] ;;
```

errore di tipo

liste di liste di interi

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

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

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

# let g n = n > 0 ;;
  g : int -> bool = <fun>

# [f ; g] ;;
  errore di tipo
```

mercoledì 20 novembre 2019 12:34

Sulle liste sono predefinite due funzioni

hd — head

tl — tail

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

hd restituisce il primo elemento di una lista. hd è indefinita su [].

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

# hd [];;
errore
```

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

tl restituisce la lista argomento senza il primo elemento. tl è indefinita su []

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

# tl [];;
errore

# hd [3;4;5] :: Tl [3;4;5];;
- : int list = [3;4;5]
```

Data una lista vogliamo scrivere una funzione CAML che restituisca l'ultimo elemento di una lista; la funzione è indefinita su liste vuote.

last [3;4;5] = 5

[5] ≡ 5::[]

```
# let rec last l =
    if tl l = [] then hd l
    else last (tl l) ;;
```

last : 'a list → 'a = <fun>

'a list: tipo di l

'a: tipo del risultato

```
last [3;4;5] ;;
= {def. last, l = [3;4;5]}
last [4;5] ;;
= {def last, l = [4;5]}
last [5] ;;
```

last [5] ;;

= { def last = l = [5] }

hd [5]

= { def . hd }

5

last [] ;;

error

Scrivere una funzione CAML che cancelli l'ultimo elemento di una lista; il risultato su [] è []

dellast [3;4;5] = [3;4]

```
let rec dellast l =
    if tl l = [] then []
      else hd l :: dellast (tl l);;
dellast : 'a list -> 'a list = <fun>
```

'a list → tipo di l

'a list → tipo risultato

```
dellast [3;4;5];;
= { def dellast, l = [3;4;5] }
hd [3;4;5] :: dellast [4;5];;
= { def hd }
  3 :: dellast [4;5];;
= { def dellast, l = [4;5] }
3 :: ( hd [4;5] :: dellast [5] )
```

$3 :: (\ hd\ [4;5] :: dellast\ [5]\ )$

$= \{ def\ hd \}$

$3 :: 4 :: dellast\ [5]$

$= \{ def\ dellast,\ l = [5] \}$

$3 :: 4 :: [\,]$

$= \{ notazione \}$

$[3;4]$