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

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

Teorema di ricorsione si applica a definizioni ricorsive di insiemi.

Data $f: \mathbb{N} \to \mathbb{N}$ il suo grafo è l'insieme F

$$F = \{ \langle n, m \rangle \mid m = f(n) \wedge n, m \in \mathbb{N} \}$$

Possiamo applicare il teorema di ricorsione alla definizione di F

Come passare dalla definizione funzionale alla definizione dell'insieme "grafico della funzione".

$$f(\underline{n}) = \begin{cases} \emptyset & \text{se } n = \emptyset \\ 2 + f(\underbrace{n-1}_{m}) & \text{se } n > \emptyset \end{cases}$$

$$F = \{\langle \emptyset, \emptyset \rangle\} \cup \{\langle n, m+2 \rangle \mid \langle n-1, m \rangle \in F\}$$

$\equiv$ avere il risultato della f per n−1 vuol dire avere in F la coppia $\langle n-1, m \rangle$

$$F = \{\langle \emptyset, \emptyset \rangle\} \cup \{\langle n+1, m+2 \rangle \mid \langle \underline{n}, \underline{m} \rangle \in F\}$$

$$F = \{\langle 0,0\rangle\} \cup \{\langle n, n+2\rangle \mid \langle n-1, n\rangle \in F\}$$

questa definizione è una eq. ricorsiva del tipo

$$X = T(X)$$

dove

$$\underline{\underline{T}}(X) = \{\langle 0,0\rangle\} \cup \{\langle n, n+2\rangle \mid \langle n-1, n\rangle \in X\}$$

$$T^0(\{\}) = \{\}$$

$$T^1(\{\}) = T(\{\}) = \{\langle 0,0\rangle\} \cup \{\langle n, n+2\rangle \mid \langle n-1, n\rangle \in \{\}\}$$

$$= \{\langle 0,0\rangle\}$$

$$T^2(\{\}) = T(\{\langle 0,0\rangle\}) =$$

$$\{\langle 0,0\rangle\} \cup \{\langle n, n+2\rangle \mid \underline{\langle n-1, n\rangle \in \{\langle 0,0\rangle\}}\}$$

$$n-1 = 0 \Rightarrow n = 1$$
$$n = 0 \Rightarrow n+2 = 2$$

$$= \{\langle 0,0\rangle, \langle 1,2\rangle\}$$

$$T^3(\{\}) = T\left(\{\langle 0,0\rangle, \langle 1,2\rangle\}\right) =$$

$$\underline{\{\langle 0,0\rangle\}} \cup \underline{\{\langle n, n+2\rangle \mid \langle n-1, n\rangle \in \{\langle 0,0\rangle, \langle 1,2\rangle\}\}}$$

$n-1 = 0 \Rightarrow n = 1$
$n = 0 \Rightarrow n = 2$

$n-1 = 1 \Rightarrow n = 2$
$n = 2 \Rightarrow n+2 = 4$

$$= \{\langle 0,0\rangle, \langle 1,2\rangle, \langle 2,4\rangle\}$$

$$T^4(\{\}) = \ldots\ldots$$

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

$$\boxed{f(n) = 2 \cdot n}$$

$$0! = 1$$

$$n! = n * (n-1) * (n-2) * \dots * 1 \qquad \text{se } n > 0$$

$$0! = 1$$

$$n! = n * (n-1)! \qquad \text{se } n > 0$$

definizione ricorsiva

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

$$Fact = \{\langle 0,1\rangle\} \cup \{\langle n, n*m\rangle \mid \langle n-1, m\rangle \in Fact\}$$

$$T(X) = \{\langle 0,1\rangle\} \cup \{\langle n, n*m\rangle \mid \langle n-1, m\rangle \in X\}$$

$$T^0(\{\}) = \{\}$$

$$T^1(\{\}) = T(\{\}) = \{\langle 0,1\rangle\} \cup \{\langle n, n*m\rangle \mid \langle n-1, m\rangle \in \{\}\}$$

$$= \{\langle 0,1\rangle\}$$

$$T^2(\{\}) = T(\{\langle 0,1\rangle\}) = \{\langle 0,1\rangle\} \cup$$

$$T(\{\}) = T(\{\langle \emptyset, 1\rangle\}) = \{\langle \emptyset, 1\rangle\} \cup$$

$$\{\langle n, n * m\rangle \mid \langle n-1, m\rangle \in \{\langle \emptyset, 1\rangle\}\}$$

$n - 1 = \emptyset \Rightarrow n = 1$

$m = 1 \Rightarrow m = 1$

$$\{\langle \emptyset, 1\rangle, \langle 1, 1\rangle\}$$

$$T^3(\{\}) = T(\{\langle \emptyset, 1\rangle, \langle 1, 1\rangle\}) = \{\langle \emptyset, 1\rangle\} \cup$$

$$\{\langle n, n * m\rangle \mid \langle n-1, m\rangle \in \{\langle \emptyset, 1\rangle, \langle 1, 1\rangle\}\}$$

$n - 1 = \emptyset \Rightarrow n = 1$

$m = 1 \Rightarrow n * m = 1$

$n - 1 = 1 \Rightarrow n = 2$

$m = 1 \Rightarrow n * m = 2$

$$= \{\langle \emptyset, 1\rangle, \langle 1, 1\rangle, \langle 2, 2\rangle\}$$

$$T^4(\{\}) = \{\langle \emptyset, 1\rangle, \langle 1, 1\rangle, \langle 2, 2\rangle, \langle 3, 6\rangle\}$$

$\vdots$

mercoledì 21 novembre 2018 11:43

CAML

→ Meta Language

Veniva usato per descrivere la semantica (significato) di programmi scritti in altri linguaggi.

ML era molto semplice

CAML light

(programmazione II)

Versione di base scaricabile

mercoledì 21 novembre 2018 11:48

3;;

- : int = 3

Valore dell'espressione

tipo del risultato

il nome che ho dato a un valore.

Non ho dato nessun nome.

INFERENZA DI TIPO

3.5;;

- : float = 3.5

3+7;;

- : int = 10

Per associare un nome a un valore si usa l'espressione **let**

```
# let n = 3+7;;
```

nome

espressione

```
n : int = 10
```

```
# n;;
- : int = 10
```

```
let f n = n+2;;
```

≡ let f (n) =

nome di funzione

nome parametro

+ somma tra interi

+. somma tra float

```
f : int -> int = <fun>
```

in molti casi non è decidibile quale funzione stiamo definendo (teoria della calcolabilità)

```
f 2;;
- : int = 4
```

mercoledì 21 novembre 2018 11:59

let $f\ n = n+2;;$

parametro

definizione della funzione

$f\ 2;;$

argomento

applicazione (o chiamata) della funzione f

CAML light
ha la stessa potenza dei linguaggi imperativi

MICROSOFT
F#

Abbiamo definito un interprete (programma in grado di eseguire) per programmi imperativi

```
# let f (n, m) = n + m + 3;;
f : int * int -> int = <fun>
```

prodotto cartesiano tra insiemi

$$A \times B = \{ \langle a, b \rangle \mid a \in A \wedge b \in B \}$$

```
# f (3,5);;
- : int = 11
```

```
# let f n m = n + m + 3;;
```

la funzione f prende un primo argomento (n) e restituisce una funzione che prende un argomento (m) e mi dà il risultato

```
f : int -> (int -> int) = <fun>
```

f : int -> (int -> int) = <fun>

CAML non le mette ma
il significato è questo

Le funzioni con argomenti in sequenza
si chiamano CURRYed.

Curry matematico americano del '900

```
# f 3 4 ;;
- : int = 10
```

```
# f 3;;
- : int -> int = <fun>
```

```
# let g = f 3 ;;
g : int -> int = <fun>
```

```
# g 4 ;;
- : int = 10
```

$f\ n\ m = n + m + 3$

$g = f\ 3 \longrightarrow 3 + m + 3$

$$\boxed{g\ x = 3 + x + 3}$$

```
let g = f 3;;
g : int -> int = <fun>
```

Come trasformare una funzione nella sua versione Curryed

```
# let t (n,m) = 2*n + m ;;
t : int * int -> int = <fun>
```

```
# t 3;;
```

errore di tipo

```
# let s n m = t (n,m);;
s : int -> (int -> int) = <fun>
```

s è la versione Curryed della funzione t

```
# s 3 4;;
- : int = 10

# t (3,4);;
- : int = 10
```

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

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

$$\# \; s \; (\underbrace{t \; (3,4)}) \; 5 \;;;$$

- : int = 25

$$s \; \overbrace{(t \; (3,4))} \; 5 \;;;$$

l'applicazione di funzioni ha precedenza sulle altre operazioni e

associa a sinistra

(s t) (3,4) 5;;

```
# let f n = n + 3
f : int -> int = <fun>
```

$\# (f\ 3) - 1;;$ $\quad$ $\# f\ (3-1);;$

mercoledì 21 novembre 2018 12:48

Curryed

~~2 m~~

```
# let g n m = 2*n + m;
g : int -> (int -> int) = <fun>

# g 3 ;;
- : int -> int = <fun>
```

```
let s = g 3 ;;
s : int -> int = <fun>
```

le funzioni, quando applicate, possono dare come risultato altre funzioni.

let f n = n ;;

f : 'a → 'a = (fun)

funzioni POLIMORFE

possono essere applicate a valori di tipo diverso.

'a è una variabile di tipo

f 3;;

- : int = 3

f "abc" ;;

- : string = "abc"

---

let f (n, m) = (n, m);;

f : 'a * 'b → 'a * 'b = (fun)

f (3, 4.5) ;;

- : int * float = (3, 4.5)

```
# let g (n, m) = (n, m+1);;
g : 'a * int -> 'a * int

# g (4.5, 3);;
- : float * int = (4.5, 4)

# g (4.5, 3.5);;
errore di tipo

# let f n = n;;
f : 'a -> 'a = <fun>

# f (3, 4.5);;
- : int * float = (3, 4.5)
```