Inferenza di tipi

Titolo nota 23/11/2015

Senza dire esplicitamente che tipo hanno le espressioni, CAML è in grado di capirlo dei VALORI e OPERAZIONI coinvolte.

Inferenza molto semplice → Inferenza complessa (quelle che coinvolgono <u>DEFINIZIONI DI FUNZIONI</u>)

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

$f(n) = n+1$

non scrive il valore della funzione perché <u>in generale</u> non è possibile dire quale funzione stiamo definendo.

TEORIA DELLA CALCOLABILITA'

(teoria fondamentale dell'informatica)

Quali funzioni sono calcolabili (per le quali esiste un algoritmo (programma)).

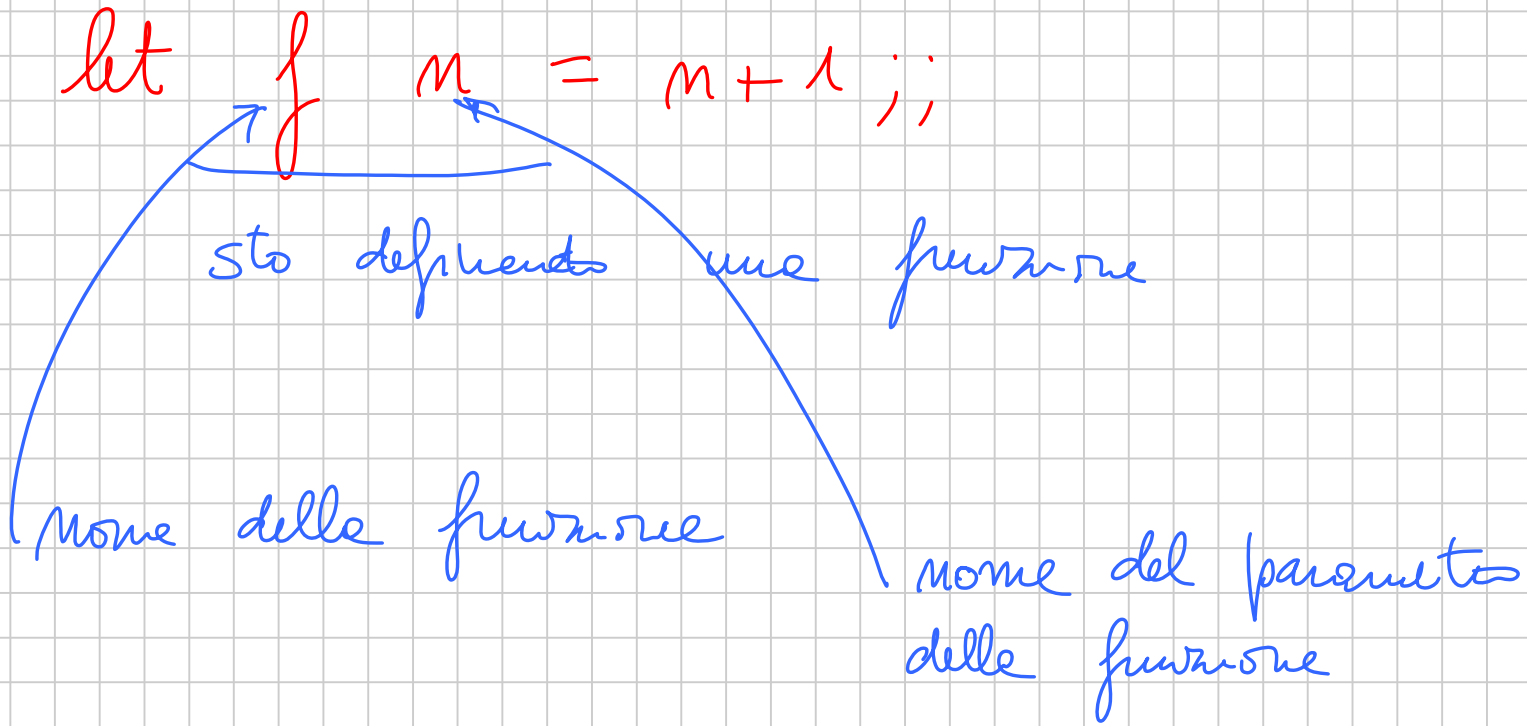
- Date due definizioni di funzioni è, in generale, indecidibile ("non si può decidere") se le due definizioni calcolano la "STESSA FUNZIONE".

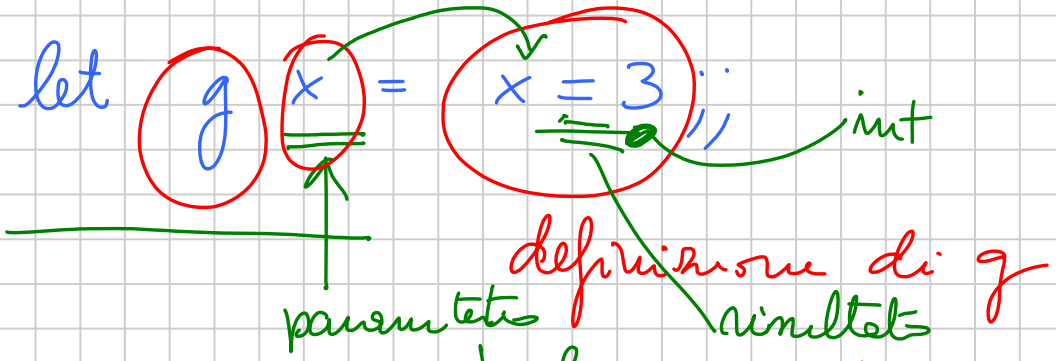
$f(n) = n+1$

$g(n) = n+2-1$

$\langle f_{un} \rangle$

Valore funzionale


let f n = n + 1 ;;
sto definendo una funzione
nome della funzione
nome del parametro della funzione



g : int → bool = <fun>

tipo dell'argomento

tipo del risultato

let f (n , m) = n + m + 1 ;;

nome della funzione

f : int * int → int = <fun>

```
let g (n , m) = n = m+1 ;;
```

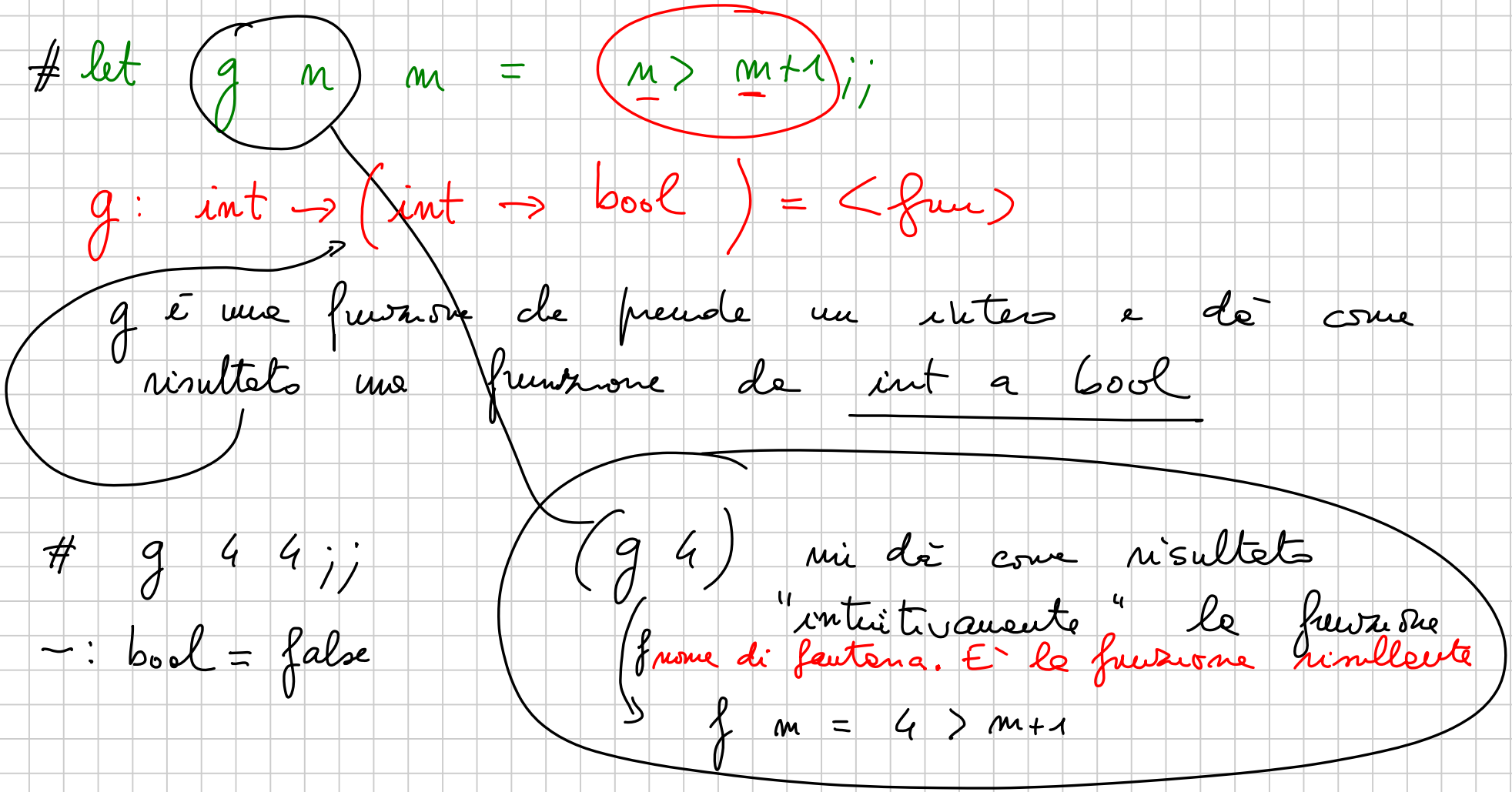
```
g : int * int -> bool = <fun>
```

tipo degli argomenti

tipo risultato

```
let ((g n) m) = n > m+1 ;;
```

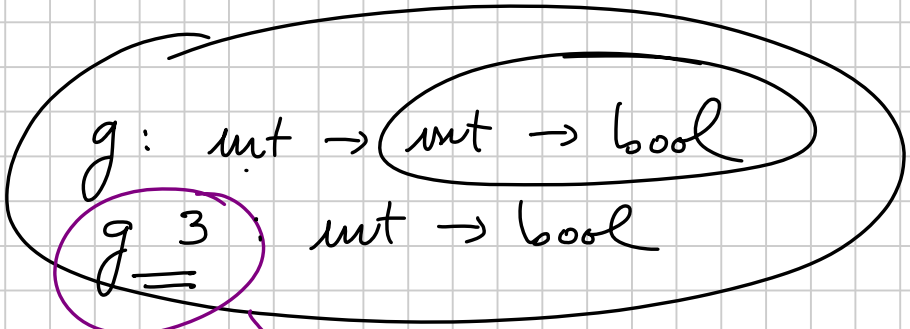
g come una funzione che si applica a n e dà come risultato una FUNZIONE che si applica a m e si ottiene il risultato dato da $n > m+1$


let g n m = n > m+1 ;;
g : int -> (int -> bool) = <fun>
g è una funzione che prende un intero e dà come risultato una funzione da int a bool
g 4 4 ;;
- : bool = false
(g 4) mi dà come risultato "intuitivamente" la funzione
f m = 4 > m+1

```
# let g n m = n > m+1 ;;
g : int -> int -> bool = <fun>

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

# f 2 ;;
- : bool = false
```



3 > 2+1

g 3 = t

t m = 3 > m+1

```
let g n m = n > m+1 ;;
```

Gli argomenti in "SEQUENZA" si chiamano

Curryed

Curry. matematico americano e logico del 1900

```
g : int -> int -> bool = <fun>
```

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

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

```
# g 3 4 ;;
- : bool = false
```

```
# let f x = x + 2 ;;
```

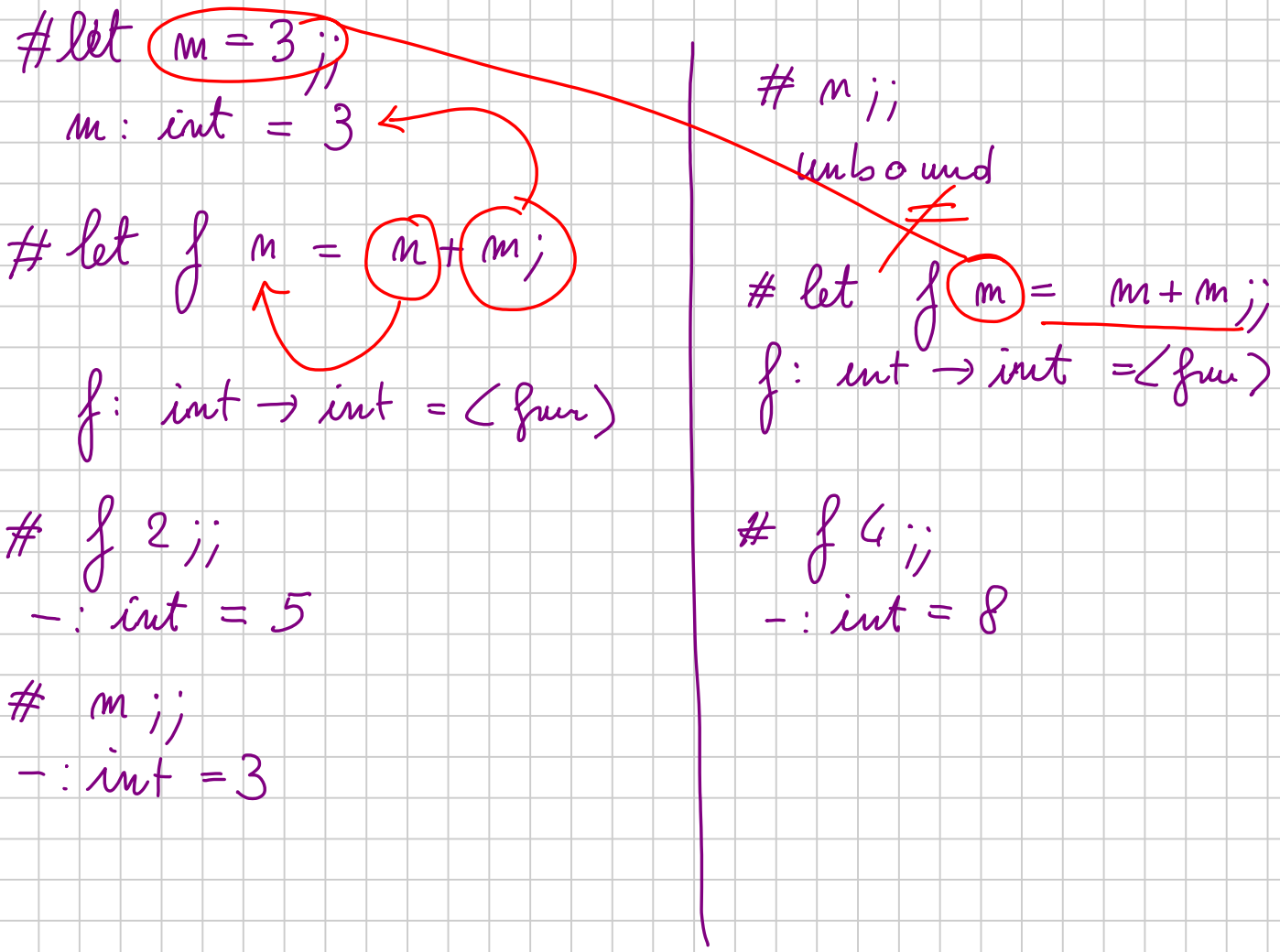
↑ parametro che è locale alla funzione

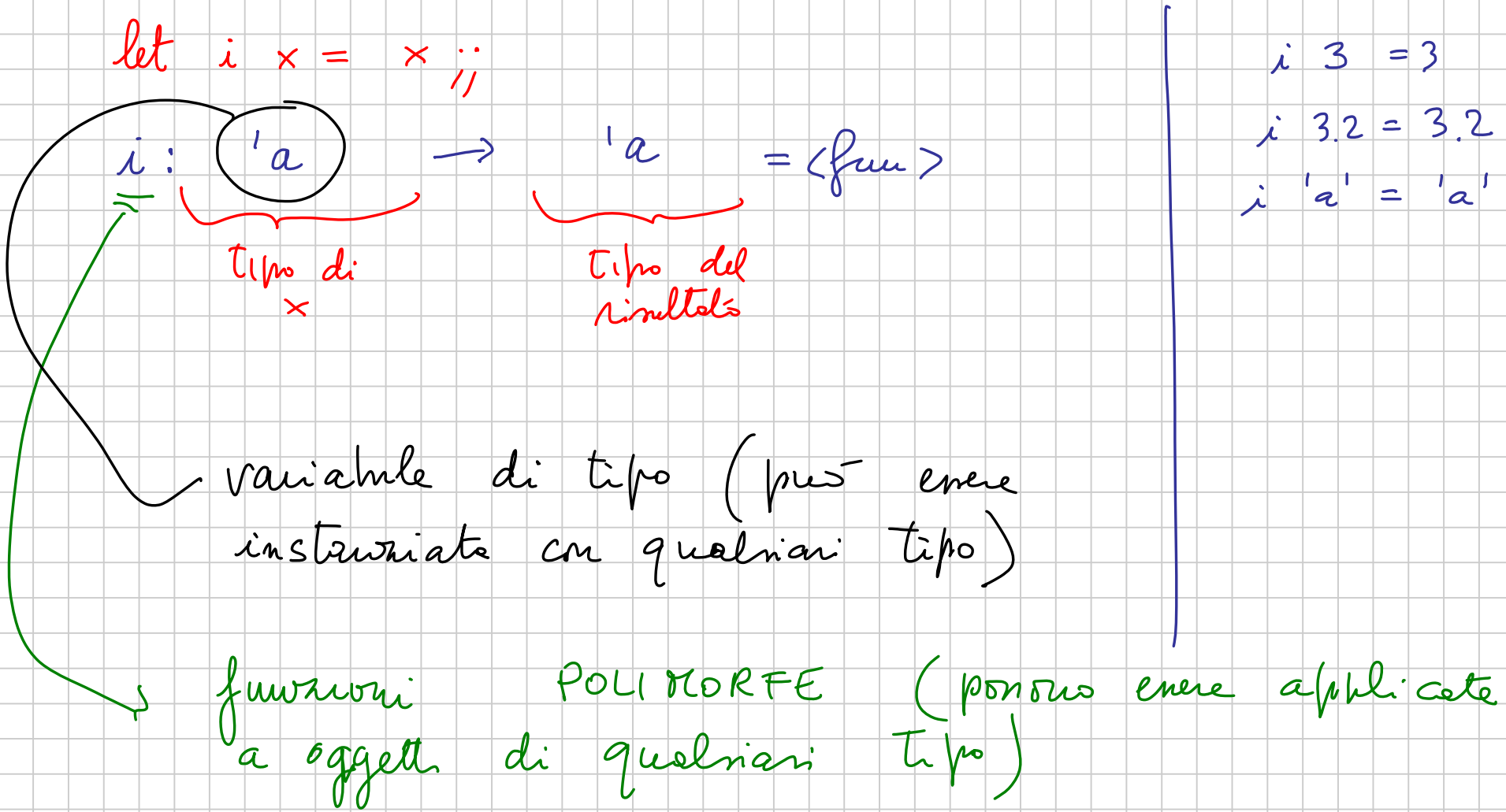
```
f : int -> int = <fun>

# x ;;
  Unbound
```

x non ha associato un valore

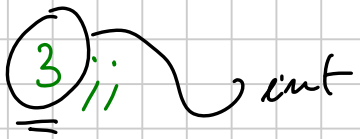
la definizione del parametro x è conosciuta solamente nella definizione di f (non fuori)


#let m = 3;;
m : int = 3
#let f n = n + m;;
f : int → int = <fun>
f 2;;
- : int = 5
m;;
- : int = 3
m;;
unbound
#let f m = m + m;;
f : int → int = <fun>
f 4;;
- : int = 8


let i x = x ;;
i : 'a → 'a = <fun>
tipo di x
tipo del risultato
variabile di tipo (può essere instanziata con qualsiasi tipo)
funzioni a oggetti
POLIMORFE
(possono essere applicate di qualsiasi tipo)
i 3 = 3
i 3.2 = 3.2
i 'a' = 'a'

```
# let i x = x;
  i : 'a -> 'a = <fun>
```

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



per applicare i a 3 si instanzia la variabile 'a a "int" e il tipo di i diventa " int → int "

```
# i 'a';;
- : char = 'a'
```

la variabile 'a nel tipo di i viene instanziata a "char" e il tipo di i diventa " char → char "

```
# let apply f x = f x ;;
```

apply : ('a → 'b) → 'a → 'b = <fun>

('a → 'b): tipo di f

'a: tipo di x

'b: tipo risultato

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

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

```
# apply g 3 ;;
- : int = 4
```

int → int

apply : (int → int) → int → int

```
# apply g true ;;
```

ERRORE DI TIPO bool

apply : (int → int) → int → int

bool