

# CONTROLLO DEI TIPI

## Importanza dei tipi:

Visto che nel  $\lambda$ -calcolo i programmi e i valori sono funzioni, possono essere commessi errori rispetto all'uso inteso dei valori.

$\text{False } 0 = (\lambda t. \lambda f. f)(\lambda s. \lambda z. z) \rightarrow \text{False si può applicare a } 0 \text{ e produce un valore che è sbagliato, false è booleano}$

I valori sono tutti rappresentati allo stesso modo ma non si possono creare tutti allo stesso modo. Questo problema è affrontato coi "Tipi".

## Tipo

- È un attributo di un dato per controllare se viene usato in modo corretto
- Limita i valori che una variabile o una funzione può assumere
- Definisce le operazioni che possono essere fatte sui dati

## Sistema di Tipi

È un metodo:

- **Sintattico**: lavora sulla sintassi, non esegue il programma
- **Effettivo**: definisce un algoritmo che controlla i vincoli sui tipi (**compilatore/interprete**)
- **Strutturale**: implementato in modo ricorsivo per dimostrare l'assenza di comportamenti anomali, strutturando le operazioni in base ai tipi di valori calcolati (il tipo di un'espressione dipende dal tipo delle sottospressioni)

## Type Safety

Se possiamo dare un tipo a un'espressione, possiamo usare il suo tipo per assicurarsi che venga usata nel modo corretto

## Controllo dei tipi

- . **Statico**: in fase di compilazione
- . **Dinamico**: in fase di esecuzione

$\Rightarrow$  Se un programma supera il controllo dei tipi è garantito che si comporti bene rispetto ai tipi

## Sintassi

- .  $E ::= \text{true} \mid \text{false} \mid \text{NV} \mid \text{if } E \text{ then } E_1 \text{ else } E_2 \mid \text{succ } E \mid \text{pred } E \mid \text{isZero } E$
- .  $V ::= \text{true} \mid \text{false} \mid \text{NV}$
- .  $NV ::= 0 \mid 1 \mid 2 \mid \dots$

## Regole di Valutazione Semantica

.  $\text{if true then } E_1 \text{ else } E_2 \longrightarrow E_1$

.  $\text{if false then } E_1 \text{ else } E_2 \longrightarrow E_2$

. 
$$\frac{E \rightarrow E'}{\text{if } E \text{ then } E_1 \text{ else } E_2 \longrightarrow \text{if } E' \text{ then } E_1 \text{ else } E_2} \rightarrow \begin{array}{l} \text{esegue finché la guardia non} \\ \text{diventa true o false} \end{array}$$

. 
$$\frac{E \rightarrow E'}{\text{succ } E \rightarrow \text{succ } E'} \quad \frac{m = n+1}{\text{succ } n \rightarrow m}$$

. 
$$\frac{E \rightarrow E'}{\text{pred } E \rightarrow \text{pred } E'} \quad \frac{n > 0 \quad m = n-1}{\text{pred } n \rightarrow m}$$

.  $\text{pred } 0 \rightarrow 0, \text{ isZero } 0 \rightarrow \text{True}$

. 
$$\frac{E \rightarrow E' \quad n > 0}{\text{isZero } E \rightarrow \text{isZero } E'} \quad \frac{}{\text{isZero } n \rightarrow \text{false}}$$

Tipi di espressioni:

$T ::= \text{Bool} \mid \text{Nat}$

Type checker

Relazione binaria tra tipo  $T$  ed espressione  $E \Rightarrow E : T$

Regole definite per induzione strutturale sulla sintassi del programma

Regole per il controllo dei tipi:

Se un'espressione non soddisfa nessuna regola, allora non ha tipo  
Un'espressione puo' avere al massimo un tipo

.  $\text{true} : \text{Bool}$ ,  $\text{false} : \text{Bool}$ ,  $n : \text{Nat}$  ( $n = 1, 2, 3, \dots$ )

.  $\frac{E : \text{Nat}}{\text{succ } E : \text{Nat}}$ ,  $\frac{E : \text{Nat}}{\text{pred } E : \text{Nat}}$ ,  $\frac{E : \text{Nat}}{\text{isZero } E : \text{Bool}}$

.  $\frac{E_1 : \text{Bool} \quad E_1 : T \quad E_2 : T}{\text{if } E_1 \text{ then } E_1 \text{ else } E_2 : T}$  l'uso di "T" garantisce la composizionalità  
(spiegata sotto)

es. albero di derivazione

$\frac{0 : \text{Nat}}{\text{isZero } 0 : \text{Bool}}$     $\frac{2 : \text{Nat}}{\text{pred } 2 : \text{Nat}}$     $\frac{1 : \text{Nat}}{\text{pred } 1 : \text{Nat}}$   
if isZero 0 then 2 else pred 1 : Nat ↑ L'espressione è ben tipata  
ed ha tipo Nat

Algoritmo Type Checker

$\text{typeof}(E) =$   
if ( $E = \text{true}$ ) then  $\text{Bool}$   
else if ( $E = \text{false}$ ) then  $\text{Bool}$   
else if ( $E = \text{if } E_1 \text{ then } E_2 \text{ else } E_3$ ) then {  
    if ( $\text{typeof}(E_1) = \text{Bool}$  and  $\text{typeof}(E_2) = \text{typeof}(E_3)$ ) then  $\text{typeof}(E_2)$   
    else "type error"  
    else if ( $E = n$ ) then  $\text{Nat}$   
    else if ( $E = \text{succ } E_1$ ) then {  
        if ( $\text{typeof}(E_1) = \text{Nat}$ ) then  $\text{Nat}$   
        else "type error"  
    else if ( $E = \text{pred } E_1$ ) then {  
        if ( $\text{typeof}(E_1) = \text{Nat}$ ) then  $\text{Nat}$   
        else "type error"  
    else if ( $E = \text{isZero } E_1$ ) then {  
        if ( $\text{typeof}(E_1) = \text{Nat}$ ) then  $\text{Bool}$   
        else "type error"  
    }

Composizionalità

Il tipo di un'espressione complessa dipende solo dalla sua struttura sintattica e dai tipi delle sue sotto-espressioni:  
es. "if" richiede che i ram: then ed else abbiano lo stesso tipo

Correttezza del sistema di tipi

E' espressa dalle due seguenti proprietà che garantiscono che l'esecuzione non si blocca a run-time durante la valutazione

. Progresso: se  $E : T$ , allora  $E$  è un valore oppure  $E \rightarrow E'$  per qualche espressione  $E'$  (puo' fare un passo)

. Conservazione: se  $E : T$  e  $E \rightarrow E'$ , allora  $E' : T$   
(il tipo è conservato:  $\text{succ}(\text{succ}(1)) : \text{Nat}$ ,  $\text{succ}(2) : \text{Nat}$ ,  $3 : \text{Nat}$ )

Nota:  $E : T$  significa che  $E$  è ben tipato.

## Lambda calcolo tipato

Sintassi  $V ::= \text{true} \mid \text{false} \mid \text{fun } x : \tau = e$   
 $e ::= x \mid \text{fun } x : \tau = e \mid \text{Apply}(e, e) \mid \text{true} \mid \text{false}$   
 $\quad \mid \text{if } e \text{ then } e \text{ else } e$

Regole di controllo dei tipi (type checker)

$$\Gamma \vdash \text{true} : \text{Bool} \quad \Gamma \vdash \text{false} : \text{Bool} \quad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash E : \text{Bool} \quad \Gamma \vdash E_1 : \tau \quad \Gamma \vdash E_2 : \tau}{\Gamma \vdash \text{if } E \text{ then } E_1 \text{ else } E_2 : \tau} \xrightarrow{\text{stesso tipo}}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau_1 \rightarrow \tau_2}{\Gamma \vdash \text{fun } x : \tau_1 = e : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash \text{Apply}(e_1, e_2) : \tau_2}$$

Tipi  $\tau ::= \text{Bool} \mid \tau \rightarrow \tau$

Semantica (valutazione)

$$\text{if true then } e_1 \text{ else } e_2 \rightarrow e_1$$

$$\text{if false then } e_1 \text{ else } e_2 \rightarrow e_2$$

$$e_1 \rightarrow e_4$$

$$\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \rightarrow \text{if } e_4 \text{ then } e_2 \text{ else } e_3$$

$$\text{Apply}(\text{fun } x : \tau = e_1, v) \rightarrow e_1\{x := v\}$$

$$e_1 \rightarrow e'$$

$$\text{Apply}(e_1, e_2) \rightarrow \text{Apply}(e', e_2)$$

$$e_2 \rightarrow e'$$

$$\text{Apply}(e_1, e_2) \rightarrow \text{Apply}(v, e_2)$$

$\beta$ -riduzione  
CBV

## Ambiente di tipo $\Gamma$

- Contiene il tipo per ogni variabile dichiarata  $\Gamma = x_1 : \tau_1, x_2 : \tau_2, \dots, x_k : \tau_k$
- Per indicare la funzione  $\Gamma(x) = \tau$
- $\Gamma, x : \tau$  è la notazione per indicare l'estensione della funzione  $\Gamma$  con l'associazione  $x : \tau$
- $\emptyset(x) = \text{undefined}$  per tutte le variabili  $x$
- Nel gergo dei compilatori è chiamata **tavella dei simboli**

## Giudizio di tipo

$\Gamma \vdash e : \tau$  indica che l'espressione  $e$  ha tipo  $\tau$  nell'ambiente di tipo  $\Gamma$

Le regole sono applicate dal compilatore in fase di analisi statica

## Tipi per le funzioni

Il costruttore  $\tau_1 \rightarrow \tau_2$  descrive il tipo delle funzioni che prendono in ingresso un argomento di tipo  $\tau_1$  e restituiscono un risultato di tipo  $\tau_2$

es.  $\emptyset, x : \text{Bool} \vdash x : \text{Bool}$  estendo l'ambiente con  $x : \text{Bool}$   
 $\emptyset \vdash \text{fun } x : \text{Bool} = x : \text{Bool} \rightarrow \text{Bool}$  prende  $x$  di tipo Bool e restituisce  $x$   
 ambiente vuoto la funzione risulta Bool

## ESERCIZIO TIPI FUNZIONI

Tipaggio corpo  
Tipaggio fun esterna  
Parte da  $\Gamma$  vuoto

$$\begin{array}{ll} \Gamma & \Gamma' \\ \emptyset, (x, \text{Bool})(x) = \text{Bool} & \emptyset(x, \text{Bool}), (y, \text{Bool})(y) : \text{Bool} \\ \emptyset, (x, \text{Bool}) \vdash x : \text{Bool} & \emptyset, (x, \text{Bool}), (y, \text{Bool}) \vdash y : \text{Bool} \\ \emptyset, (x, \text{Bool}) \vdash \text{if } x \text{ then... else...} : \text{Bool} \rightarrow \text{Bool} & \emptyset, (x, \text{Bool}) x \vdash (\text{fun } y : \text{Bool} = y) : \text{Bool} \rightarrow \text{Bool} \quad \text{Prende Bool e restituisce Bool} \\ \emptyset \vdash \text{fun } x : \text{Bool} = \text{if } x \text{ then... else...} : \text{Bool} \rightarrow (\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{su entrambi i rami:} & \\ \emptyset \vdash \text{APPLY}(\text{fun } x : \text{Bool} = \text{if } x \text{ then } (\text{fun } y : \text{Bool} = y) \text{ else } (\text{fun } y : \text{Bool} = y), \text{true}) : \text{Bool} \rightarrow \text{Bool} & \end{array}$$

\* Lo sviluppo a destra ( $\Gamma'$ ) andrebbe fatto 2 volte perché quello è il ramo "then" e manca il ramo "else", ma viene saltato visto che sono uguali.

## Type Safety

Le seguenti proprietà fanno sì che il programma non si blochi e garantiscono la correttezza del sistema di tipi:

. Progresso: se  $\emptyset \vdash e : \tau$ , allora  $e$  è un valore oppure  $e \rightarrow e'$  per qualche espr.  $e'$

. Conservazione: se  $\Gamma \vdash e : \tau$  e  $e \rightarrow e'$ , allora  $\Gamma \vdash e' : \tau$

## Substitution Lemma

I tipi sono preservati dall'operazione di sostituzione.  
Se  $\Gamma, x : \tau_1, t : \tau \vdash e : \tau$  e  $\Gamma \vdash e_1 : \tau_1$  allora  $\Gamma \vdash e[x := e_1] : \tau$

Ovvero: sostituire un parametro formale con un parametro attuale dello stesso tipo, fa sì che si preservi il tipo dell'espressione

## Estensione Linguaggio

E' possibile estendere il linguaggio con altri costrutti:

Sintassi :

$e ::=$	Espressioni
$x$	Variabili
fun $x : \tau = e$	Funzioni
Apply( $e, e$ )	Applicazione
true	Costante true
false	Costante false
$n$	Costanti numeriche
$e \oplus e$	Operazioni binarie
if $e$ then $e$ else $e$	Condizionale

$n ::= 0, 1, 2, \dots$   
 $\oplus ::= +, -, *, /$

Si assume che siano noti i tipi delle operazioni primitive  $\oplus : \tau_1 \times \tau_2 \rightarrow \tau$

Regole di Tipo:

$\Gamma \vdash n : Nat$

$$\frac{\Gamma \vdash e_1 : \tau_1, \Gamma \vdash e_2 : \tau_2}{\begin{array}{c} \oplus : \tau_1 \times \tau_2 \rightarrow \tau \\ \Gamma \vdash e_1 \oplus e_2 : \tau \end{array}}$$

## Dichiarazioni Locali:

$e ::= \dots$

$\text{let } x = e_1 \text{ in } e_2 : \tau_2$

Regole di valutazione

$$\frac{e_1 \rightarrow e'}{\text{let } x = e_1 \text{ in } e_2 : \tau_2 \rightarrow \text{let } x = e' \text{ in } e_2 : \tau_2}$$

$$\text{let } x = v \text{ in } e_2 : \tau_2 \rightarrow e_2[x := v] : \tau_2$$

$e ::= \dots$

$\text{let } x = e_1 \text{ in } e_2 : \tau_2$

Regola di tipo

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}$$

## Ricorsione

L'autoapplicazione con i combinatori:  $\Omega$  e  $\text{Y}$  non è tipabile nel  $\lambda$ -calcolo  
Dove essere esteso il linguaggio con un costrutto per la ricorsione che sia tipabile.

$G = \lambda f. \lambda n. \text{if } (n = 0) \text{ then } 1 \text{ else } n * f(n - 1)$

Fattoriale

let  $G = \text{fun } f : Nat \rightarrow Nat =$

$\text{fun } n : Nat =$

$\text{if (isZero } n) \text{ then } 1 \text{ else Times (n f (pred } n))$

$G : (Nat \rightarrow Nat) \rightarrow Nat \rightarrow Nat$

$\text{fix}(\lambda x : \tau. e) \rightarrow e[x := \text{fix}((\lambda x : \tau. e))]$

Esempio di applicazione

$\text{fact } 3 = (\text{fix } G) 3 = (\text{fix}(\lambda f. \lambda n. \text{if } (n = 0) \text{ then } 1 \text{ else } n * f(n - 1))) 3 \rightarrow$   
 $(\lambda n. \text{if } (n = 0) \text{ then } 1 \text{ else } n * f(n - 1))[f := \text{fix } G] 3 =$   
 $(\lambda n. \text{if } (n = 0) \text{ then } 1 \text{ else } n * \text{fix } G(n - 1)) 3 \rightarrow$   
 $(\text{if } (3 = 0) \text{ then } 1 \text{ else } 3 * \text{fix } G(3 - 1)) \rightarrow^* 3 * \text{fix } G(3 - 1)$