

# Algoritmo di Inferenza

Un algoritmo per decidere se una  $\lambda$ -espressione è ben tipata e per inferirne il tipo automaticamente.

E' diviso in 3 fasi:

1. **Costruzione** dell'**albero sintattico** della  $\lambda$ -espressione

2. **Annotazione** dell'albero e **generazione** dei vincoli

- **variabili di tipo:** tipo sconosciuto
- **espressioni di tipo:** tipo (parzialmente) sconosciuto
- **vincolo:** relazione di uguaglianza tra tipi espressa nella regola di tipo

3. **Risoluzione** dei vincoli

- Determinare se il sistema ammette almeno una **soluzione**
- Calcolare la soluzione più **generale**, da cui derivare tutte le altre

## Costruzione dell'albero sintattico

Prima di tutto, rappresentiamo la  $\lambda$ -espressione come un **albero**.

I nodi **interni** e le **foglie** vengono opportunamente etichettati.

Inoltre indichiamo con  $T[M]$  l'albero corrispondente alla  $\lambda$ -espressione  $M$

$T[x] \stackrel{\text{def}}{=} x$  foglia

$T[\text{c}] \stackrel{\text{def}}{=} \text{c}$  foglia

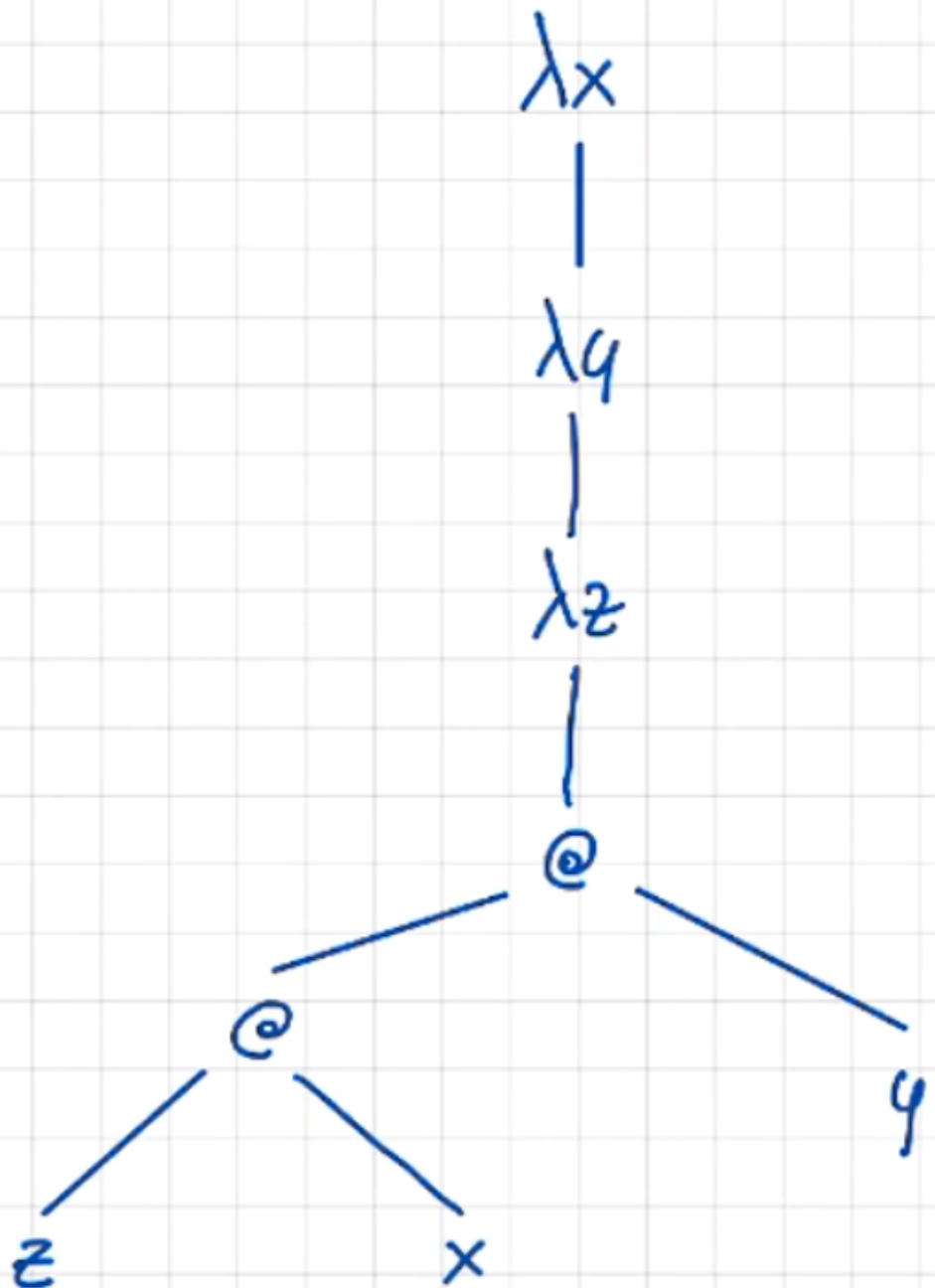
$T[\lambda x.M] \stackrel{\text{def}}{=} \begin{array}{c} \lambda x \\ | \\ T[M] \end{array}$

$T[M\ N] \stackrel{\text{def}}{=} \begin{array}{c} @ \\ / \quad \backslash \\ T[M] \quad T[N] \end{array}$

$T[\text{if } M\ N_1\ N_2] \stackrel{\text{def}}{=} \begin{array}{c} \text{if} \\ / \quad | \quad \backslash \\ T[M] \quad T[N_1] \quad T[N_2] \end{array}$

Esempio di albero sintattico:

$\lambda x. \lambda y. \lambda z. z \ x \ y$



"@" è l'operatore invisibile di applicazione

## Annotazione e generazione dei vincoli

Ogni nodo dell'albero viene etichettato con una **espressione di tipo** utilizzando una strategia **bottom-up** (dalle foglie verso la radice.)

### Espressioni di tipo

Definiamo un insieme  $TVar = \{\alpha, \beta, \gamma, \dots\}$  infinito di **variabili di tipo**;  $\alpha$  rappresenta un tipo sconosciuto, ancora da determinare.

# Sintassi espressioni di tipo

Espressioni di tipo	$\tau, \sigma$	$::=$	$\alpha$	variabile di tipo
			$\text{Bool}$	booleani
			$\tau \rightarrow \sigma$	funzioni

Un **vincolo** è una coppia di espressioni di tipo scritta come

$\tau = \sigma$

## Generazione dei vincoli

Per annotare e generare i vincoli, guardiamo le **regole di tipo(!!!)**, ricordandosi di usare una strategia **BOTTOM-UP!!**

Albero	Note
$x : \alpha$	$\alpha$ è nuova, avendo cura di usare la stessa $\alpha$ per tutte le occorrenze della stessa $x$
$c : \text{Bool}$	Altre costanti richiederanno tipi diversi

$\lambda x : \alpha \rightarrow \tau$   $T : \tau$	$\alpha$ è la variabile di tipo usata per annotare $x$ in $T$ se $x$ compare in $T$ o è nuova altrimenti
--	--

Procedendo dal basso, annoto l'albero  $T$ (corpo dell'astrazione) con il tipo  $\tau$ .

Salendo, incontro l'astrazione che ha tipo della forma  $t \rightarrow s$ .

In questo caso, il codominio è  $\tau$  stesso, mentre il dominio è  $\alpha$ .

Quindi in sintesi, il tipo di un'astrazione( $\lambda x. T$ ) è dato da  $t \rightarrow s$  dove  $t$  è il tipo dell'argomento( $\lambda x$ ) e  $s$  il tipo del corpo( $T$ )

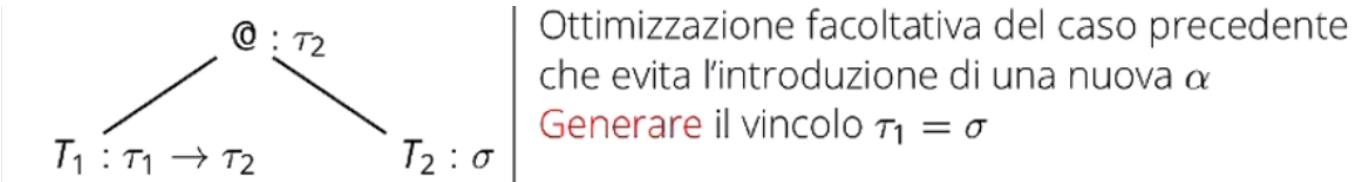
$\begin{matrix} & @ : \alpha & \\ / & & \backslash \\ T_1 : \tau & & T_2 : \sigma \end{matrix}$	$\alpha$ è nuova Generare il vincolo $\tau = \sigma \rightarrow \alpha$
---	--

Secondo la regola di tipo t-app, sappiamo che:

$T_1$  è una funzione e quindi ha tipo  $t \rightarrow s$

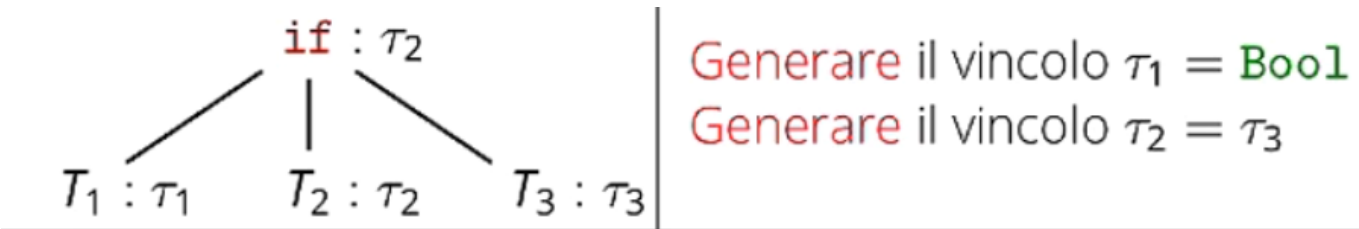
$T_2$  è l'argomento "passato" alla funzione e quindi ha tipo  $t$  che corrisponde al dominio di  $T_1$

Quindi genero il **vincolo**  $\tau = \sigma \rightarrow \alpha$  dove  $\sigma$  è il dominio di  $T_1$  che è uguale al tipo di  $T_2$  mentre  $\alpha$  è nuova



Evito l'introduzione di una variabile di tipo nuova.  
Sappiamo già che il tipo di  $T_1$  è una  $\rightarrow$  quindi possiamo subito annotare  $\tau_1 \rightarrow \tau_2$ .  
Sappiamo già inoltre il tipo di "@" che corrisponde al codominio di  $T_1$ , quindi posso annotare la radice dell'albero con  $\tau_2$ .  
Generiamo infine il **vincolo**  $\tau_1 = \sigma$  visto che il dominio di  $T_1$  deve coincidere con quello di  $\tau_2$

Possiamo applicare questa annotazione solo se sappiamo già il tipo di  $T_1$



$T_1$  deve avere tipo Bool quindi genero il **vincolo**  $\tau_1 = Bool$   
Inoltre il tipo di  $T_2$  deve essere lo stesso di  $T_3$   
quindi genero il **vincolo**  $\tau_2 = \tau_3$

$T_1$  sarebbe la condizione, mentre  $T_2$  e  $T_3$  sono gli statement

## Risoluzione dei vincoli

La fase di generazione dei vincoli genera un **sistema** della forma:  
 $\{\tau_i = \sigma_i\}_{1 \leq i \leq n}$

Bisogna determinare se tale sistema ammetta una **soluzione**.

**Definizione (sostituzione)**  
Una **sostituzione**  $\theta$  è una funzione da variabili di tipo a espressioni di tipo. Scriviamo  $\theta(\tau)$  per l'espressione ottenuta da  $\tau$  sostituendo ogni  $\alpha$  con  $\theta(\alpha)$ .

Ricordandoci che una espressione di tipo è della forma  $\tau \rightarrow \sigma$

oppure una costante Bool

Definizione (soluzione)

Dato un sistema di vincoli  $\{\tau_i = \sigma_i\}_{1 \leq i \leq n}$  e una sostituzione  $\theta$  diciamo che  $\theta$  è **soluzione** (o **unificatore**) del sistema se  $\theta(\tau_i) = \theta(\sigma_i)$  per ogni  $1 \leq i \leq n$ . Diciamo inoltre che  $\theta$  è l'**unificatore più generale** del sistema se ogni soluzione del sistema è ottenibile componendo  $\theta$  con un'altra sostituzione.

Algoritmo di risoluzione

Se c'è un vincolo	e	allora
$\tau = \tau$	—	eliminare il vincolo

Banalmente, è un vincolo sempre vero, quindi lo eliminiamo

$\tau = \alpha$	$\tau$ non è una variabile	rimpiazzare il vincolo con $\alpha = \tau$
-----------------	----------------------------	--

L'idea di fondo è di arrivare ad un sistema della forma  $\alpha_i = \tau_i$  dove  $\alpha_i$  è una variabili di tipo e  $\tau_i$  è un'espressione di tipo.

$\tau \rightarrow \tau' = \sigma \rightarrow \sigma'$	—	rimpiazzare il vincolo con $\tau = \sigma$ e $\tau' = \sigma'$
---	---	--

Dividiamo il vincolo in due.  
I due vincoli risultanti sono dati dall'uguaglianza rispettivamente dei due domini e codomini

$\tau \rightarrow \sigma = \text{Bool}$ o $\text{Bool} = \tau \rightarrow \sigma$	—	💀 l'algoritmo fallisce (type error)
--	---	-------------------------------------

Banalmente, una costante come Bool non può essere uguale ad una espressione di tipo/una funzione.

$\alpha = \tau$	$\alpha \neq \tau$ ma $\alpha$ compare in $\tau$	💀 l'algoritmo fallisce (occur check)
-----------------	---	--------------------------------------

Prendiamo il vincolo d'esempio  $\alpha = \alpha \rightarrow \beta$ . Sono chiaramente due termini diversi, ma il termine a sinistra compare anche in quello a destra, il che è ovviamente impossibile

$\alpha = \tau$	$\alpha$ non compare in $\tau$ $\alpha$ compare altrove	sostituire $\alpha$ con $\tau$ in tutti gli altri vincoli ( $\alpha = \tau$ rimane)
-----------------	--	--

Se ho due vincoli:  $\alpha = \beta$  e  $\beta = \gamma$ , applicando questa trasformazione avrò:  $\alpha = \gamma$  e  $\beta = \gamma$  (quest'ultimo vincolo rimane)

Quando nessuna trasformazione diventa applicabile, l'algoritmo termina con **successo**.

Dunque, applicando l'algoritmo ad un sistema di vincoli abbiamo che:

- 1 Prima o poi l'algoritmo fallisce o ha successo.
- 2 Se l'algoritmo fallisce, allora il sistema iniziale è insoddisfacibile.
- 3 Se l'algoritmo ha successo, allora:
  - il sistema finale ha la forma  $\{\alpha_i = \rho_i\}_{1 \leq i \leq m}$  in cui ciascuna  $\alpha_i$  compare una sola volta nel sistema
  - la sostituzione  $\theta \stackrel{\text{def}}{=} \{\alpha_i \mapsto \rho_i\}_{1 \leq i \leq m}$  è l'unificatore più generale del sistema iniziale, in particolare  $\theta(\tau_i) = \theta(\sigma_i)$  per ogni  $1 \leq i \leq n$

## Estensioni

### Numeri interi

Aggiungiamo alle costanti i **numeri interi**:

$c \in \{\text{FALSE}, \text{TRUE}, 0, 1, \dots\}$

e aggiungiamo il tipo `Int` anche alle espressioni di tipo

$\tau, \sigma ::= \dots \mid \text{Int}$

Per le **fasi 1 e 2** dell'algoritmo, non cambia niente

Per la **fase 3** invece aggiungiamo un'altra condizione di errore:

Se c'è un vincolo  $\tau \rightarrow \sigma = \text{Int}$  o  $\text{Int} = \tau \rightarrow \sigma$  o  $\text{Int} = \text{Bool}$  o  $\text{Bool} = \text{Int}$  l'algoritmo fallisce (type error)

### Liste

Aggiungiamo alle costanti i **costruttori canonici di liste** :

$c \in \{\dots, [], (:)\}$

e aggiungiamo il tipo *lista* anche alle espressioni di tipo

$\tau, \sigma ::= \dots \mid [\tau]$

Per la **fase 1** dell'algoritmo, non cambia niente

Per la **fase 2** **ogni** occorrenza di un costruttore fa uso di **nuove** variabili di tipo

Per la **fase 3** aggiungo un'altra condizione di errore

Se c'è un vincolo  $[\tau] = \text{Bool}$  o  $\text{Bool} = [\tau]$  o  $[\tau] \circ [\tau] = \sigma_1 \rightarrow \sigma_2$  o ... l'algoritmo fallisce (type error)

## Funzioni di libreria

Aggiungiamo alle costanti le **funzioni di libreria**:

$c \in \{\dots, \text{id}, \text{head}, \text{tail}, \dots\}$

Nessuna variazione per la **fase 1**.

Per la **fase 2** invece **ogni** occorrenza di una funzione di libreria fa uso di nuove variabili di tipo

Nessuna variazione per la **fase 3**

## Definizioni ricorsive

$f = M$  dove  $f$  può comparire in  $M$

**Fase 1:** il nome  $f$  è trattato come ogni altra variabile

**Fase 2:** il nome  $f$  è trattato come ogni altra variabile, inoltre alla fine dell'annotazione, generare il vincolo  $\alpha = \tau$  dove  $\alpha$  è la variabile di tipo associata a  $f$  mentre  $\tau$  è l'annotazione di  $M$

**Fase 3:** nessuna variazione