Algoritmo di Inferenza

Un algoritmo per decidere se una λ-espressione è ben tipata e per inferirne il tipo automaticamente.
E' diviso in 3 fasi:

1. **Costruzione** dell'**albero sintattico** della λ-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 λ-espressione come un **albero.**
I nodi **interni** e le **foglie** vengono opportunamente etichettati.
Inoltre indichiamo con $T[M]$ l'albero corrispondente alla λ-espressione M

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

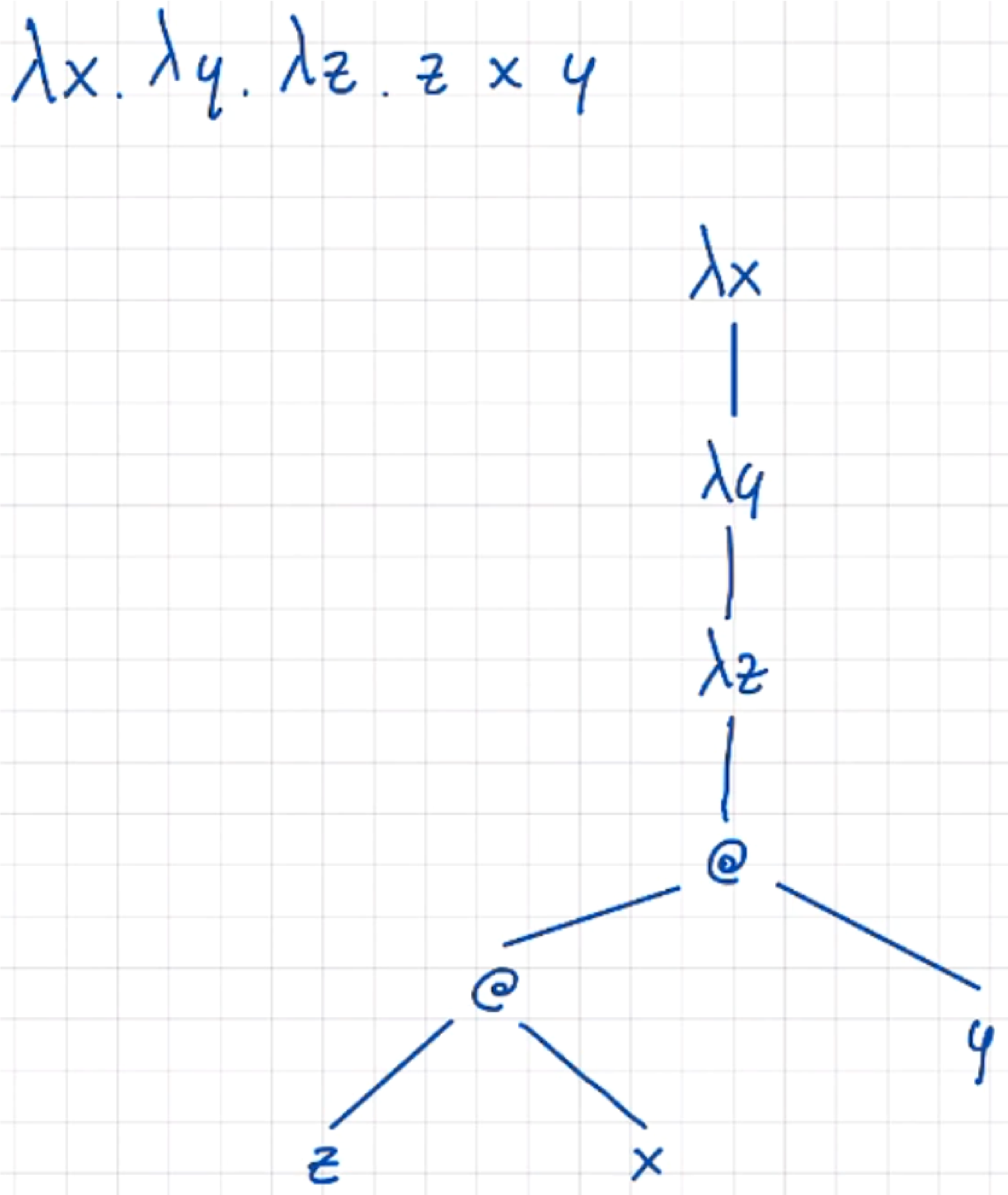
$$T[\mathtt{c}] \stackrel{\text{def}}{=} \mathtt{c} \qquad \text{foglia}$$

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

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

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

Esempio di albero sintattico:



"@" *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 $\mathrm{TVar} = \{\alpha, \beta, \gamma, \ldots\}$ infinito di **variabili di tipo**; α rappresenta un tipo sconosciuto, ancora da determinare.

Sintassi espressioni di tipo

Espressioni di tipo	τ, σ	$::=$	α	variabile di tipo
		$\mid$	$\texttt{Bool}$	booleani
		$\mid$	$\tau \to \sigma$	funzioni

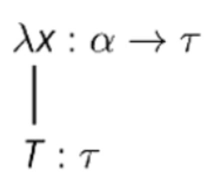
Un **vincolo** è una coppia di espressioni di tipo scritta come $\tau = \sigma$

Generazione dei vincoli

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

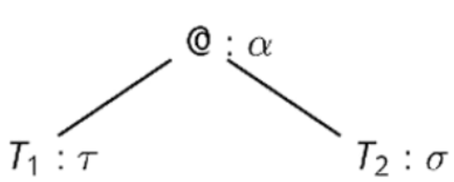
Albero	Note
$x : \alpha$	α è nuova, avendo cura di usare la stessa α per tutte le occorrenze della stessa x
$\texttt{c} : \texttt{Bool}$	Altre costanti richiederanno tipi diversi

---



α è la variabile di tipo usata per annotare x in T se x compare in T o è nuova altrimenti

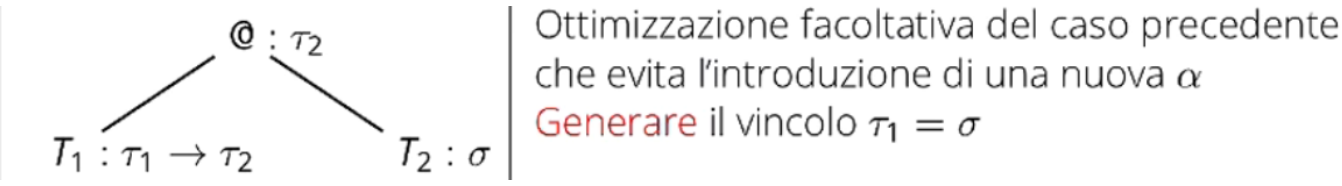
Procedendo dal basso, annoto l'albero T(corpo dell'astrazione) con il tipo τ.
Salendo, incontro l'astrazione che ha tipo della forma $t \to s$.
In questo caso, il codominio è τ stesso, mentre il dominio è α.
Quindi in sintesi, il tipo di un'astrazione($\lambda x.T$) è dato da $t \to s$ dove t è il tipo dell'argomento(λx) e s il tipo del corpo(T)



α è nuova
Generare il vincolo $\tau = \sigma \to \alpha$

Secondo la regola di tipo t-app, *sappiamo che:*
T_1 *è una funzione e quindi ha tipo* $t \to s$
T_2 *è l'argomento "passato" alla funzione e quindi ha tipo* t *che corrisponde al dominio di* T_1

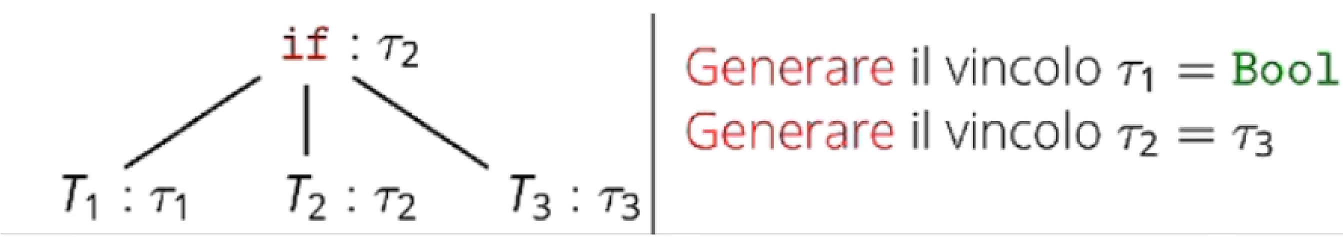
Quindi genero il ***vincolo*** $\tau = \sigma \to \alpha$ *dove* σ *è il dominio di* T_1 *che è uguale al tipo di* T_2 *mentre* α *è nuova*



Evito l'introduzione di una variabile di tipo nuova.
Sappiamo già che il tipo di T_1 *è una* $\to$ *quindi possiamo subito annotare* $\tau_1 \to \tau_2$*.*
Sappiamo già inoltre il tipo di "@" che corrisponde al codominio di T_1*, quindi posso annotare la radice dell'albero con* τ_2*.*

Generiamo infine il ***vincolo*** $\tau_1 = \sigma$ *visto che il dominio di* T_1 *deve coincidere con quello di* τ_2

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



T_1 *deve avere tipo* Bool *quindi genero il* ***vincolo*** $\tau_1 = \mathrm{Bool}$
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: $\{\tau_i = \sigma_i\}_{1 \le i \le n}$

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

Definizione (sostituzione)

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

Ricordandoci che una espressione di tipo è della forma $\tau \to \sigma$ oppure una costante Bool

Definizione (soluzione)

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

$\tau = \alpha$	τ 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 α_i è una variabili di tipo e τ_i è un'espressione di tipo.

$\tau \to \tau' = \sigma \to \sigma'$	—	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

---

$\tau \to \sigma = \texttt{Bool}$ o $\texttt{Bool} = \tau \to \sigma$	—	☠ 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 α compare in τ	☠ l'algoritmo fallisce (occur check)

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

$\alpha = \tau$	α non compare in τ α compare altrove	sostituire α con τ 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 α_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**:
$c \in \{\text{FALSE}, \text{TRUE}, 0, 1, ..\}$

e aggiungiamo il tipo Int anche alle espressioni di tipo
$\tau, \sigma ::= \ldots |\text{Int}$

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 $\tau \to \sigma = \texttt{Int}$ o $\texttt{Int} = \tau \to \sigma$ o $\texttt{Int} = \texttt{Bool}$ o $\texttt{Bool} = \texttt{Int}$ 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
$\tau, \sigma ::= \dots | [\tau]$

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 $[\tau] = \texttt{Bool}$ o $\texttt{Bool} = [\tau]$ o $[\tau] = \sigma_1 \to \sigma_2$ o $\dots$ l'algoritmo fallisce (type error)

Funzioni di libreria

Aggiungiamo alle costanti le **funzioni di libreria**:
$c \in \{\dots, \mathrm{id}, \mathrm{head}, \mathrm{tail}, ..\}$

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 α è la variabile di tipo associata a f mentre τ è l'annotazione di M
Fase 3: nessuna variazione

Nota bene: applico l'algoritmo soltanto al membro destro dell'espressione (in questo solo ad **M**)