Type inference su linguaggio base + Nat<:Bool

Simone Ballarin, approfondimento AALP 2018/19
matricola 1207245

Linguaggio Base senza subtyping

ALGORITMO DI MITCHELL WAND

Semplice algoritmo guidato dalla forma dei termini

Risaliva l’albero di derivazione con i Lemmi di Inversione, fino ad ottenere solo assiomi

Otteneva dei vincoli sul tipo dell’espressione

Risolve il sistema di vincoli con un algoritmo di unificazione

Subtyping con subsumption

Le derivazioni non sono più guidate dai termini, non si può risalire l'albero come prima. Questo vale sia per i giudizi di tipo, sia per i giudizi di subtyping.

1. T-Subsumption (si può applicare ad ogni termine sempre)
2. T-Trans (si possono usare per qualsiasi coppia di tipi S,T. Inoltre bisogna indovinare la variabile U.)
3. T-Refl (regola non strettamente necessaria, a patto di inserire NatNat e BoolBool)

Un algoritmo "bottom up" non sa' se applicare la regola guidata dalla sintassi o queste regole sempre applicabile.

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T}$$

$$\frac{S <: U \qquad U <: T}{S <: T}$$

Sottotipo algoritmico

Basato unicamento sulla struttura del tipo, quindi senza usare S-Trans e S-Refl,

avendo gli assiomi Nat<:Bool , Nat<:Nat e Bool <:Bool in aggiunta.

```
data Type =
        TBool | TNat | TArrow Type Type | Top
        deriving (Show, Eq)
```

Lemma 1: S<:S senza S-Refl

induzione strutturale su S

Lemma 2: S<:T derivabile implica S<:T derivabile senza S-Trans

induzione su S<:T

Questi due lemmi ci dicono che le due regole non guidate dalla sintassi non sono necessarie se si aggiungono NatNat e BoolBool (che sono guidate dalla sintassi).

Lemma 3: S<:T derivabile sse S<:T derivabile alg

Questo lemma combina i due precedenti formalizzando la correttezza e completezza del typing algoritmico

Correttezza codice

Lemma 4: S<:T derivabile algoritmicamente sse (<=) S T ha valore True.

Lemma 5: Se S<:T derivabile algoritmicamente allora (<=) S T=True altrimenti (<=) S T=False

Typing algoritmico

Stesso problema dell'algoritmo di sottotipaggio, siccome è presente la regola T-Sub allora l'algoritmo non risulta più guidato dalla struttura del termine.

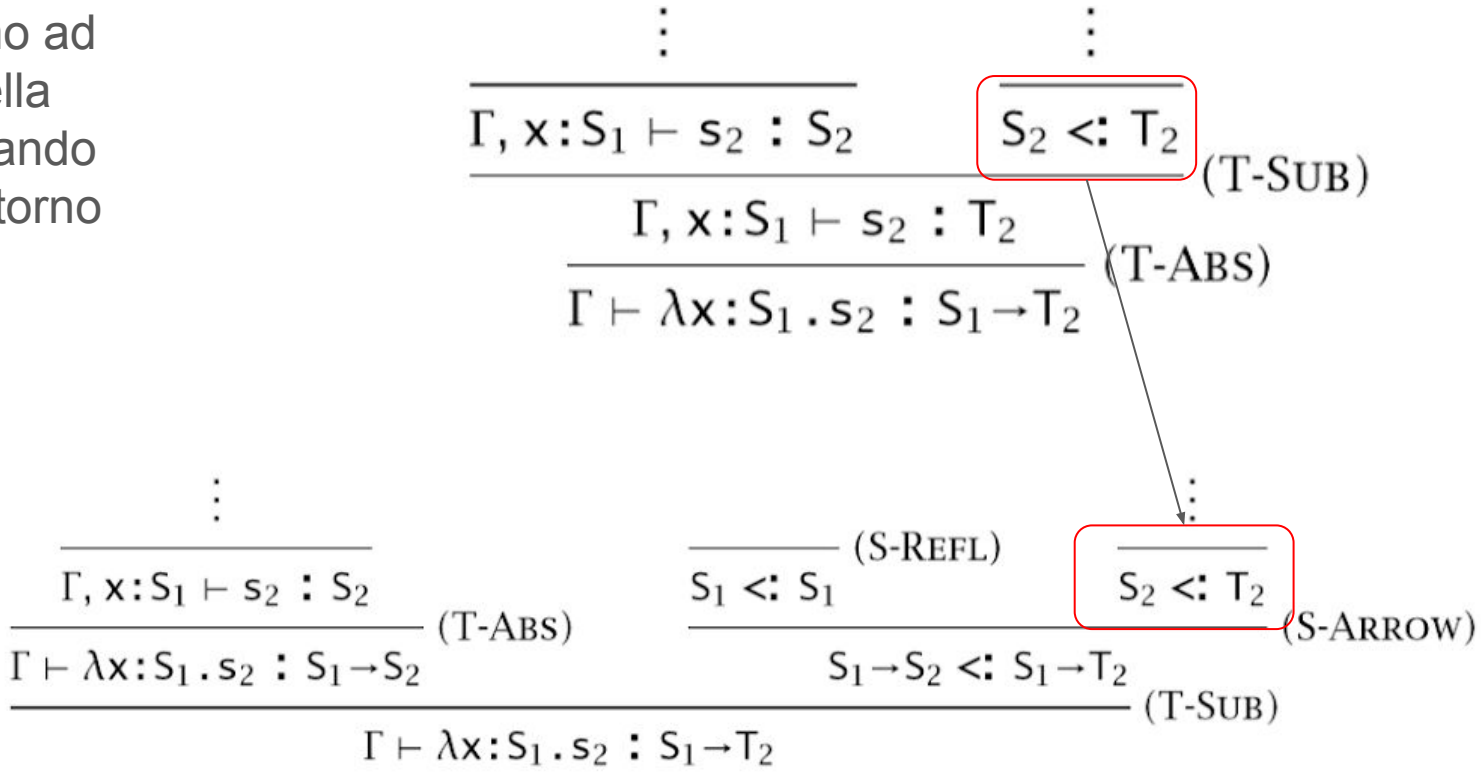
Gli unici momenti in cui T-Sub è essenziale sono:

1. quando dobbiamo tipare il parametro di una funzione con un suo sottotipo in un'applicazione. *(fn x:Bool . x) 3*
2. quando dobbiamo tipare la guardia di *if 3 then 4 else 7*

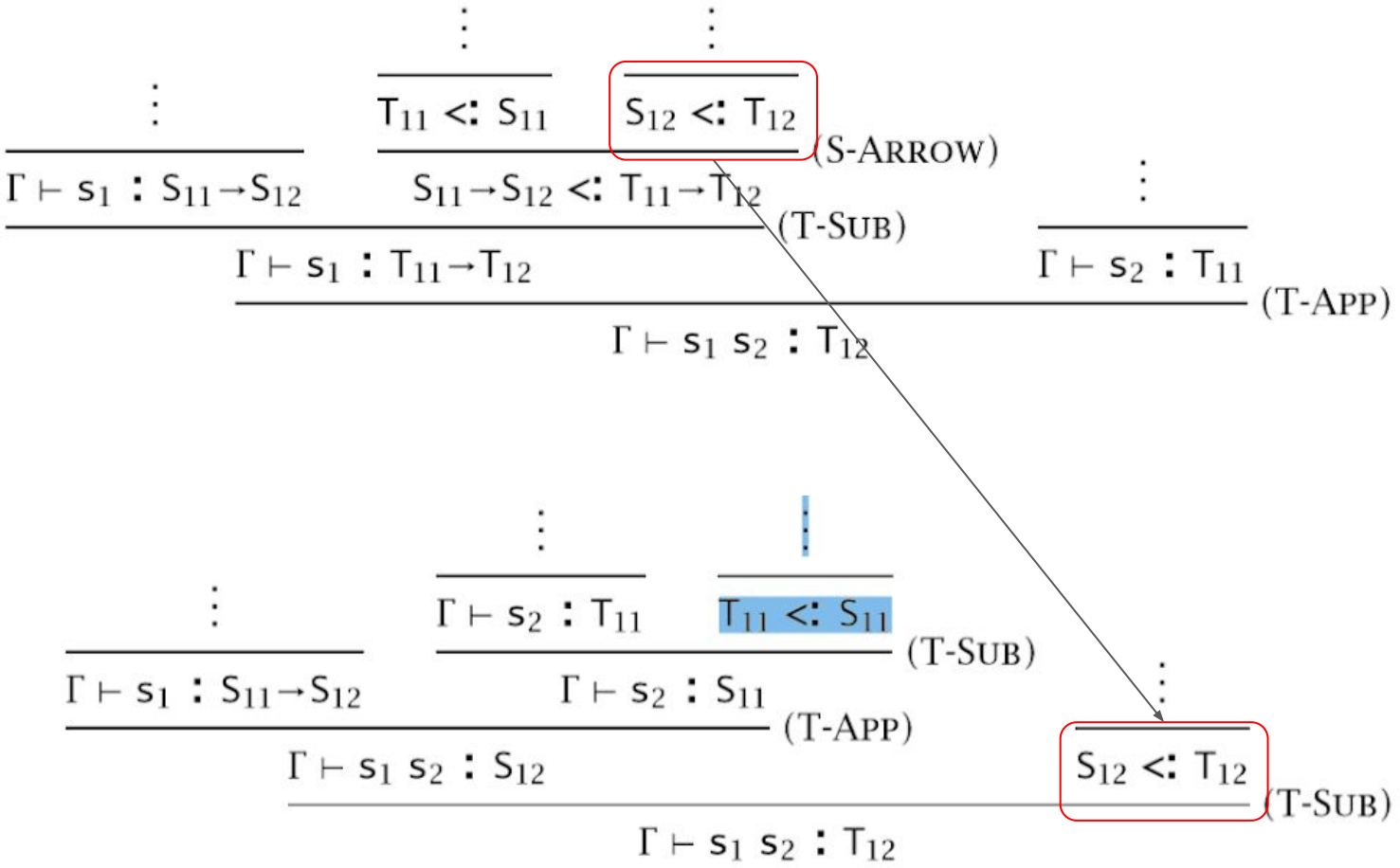
In tutti gli altri casi T-Sub può essere portato più verso la radice e quindi si può fare in modo che l'albero finisca con T-Sub.

Esempio 1:

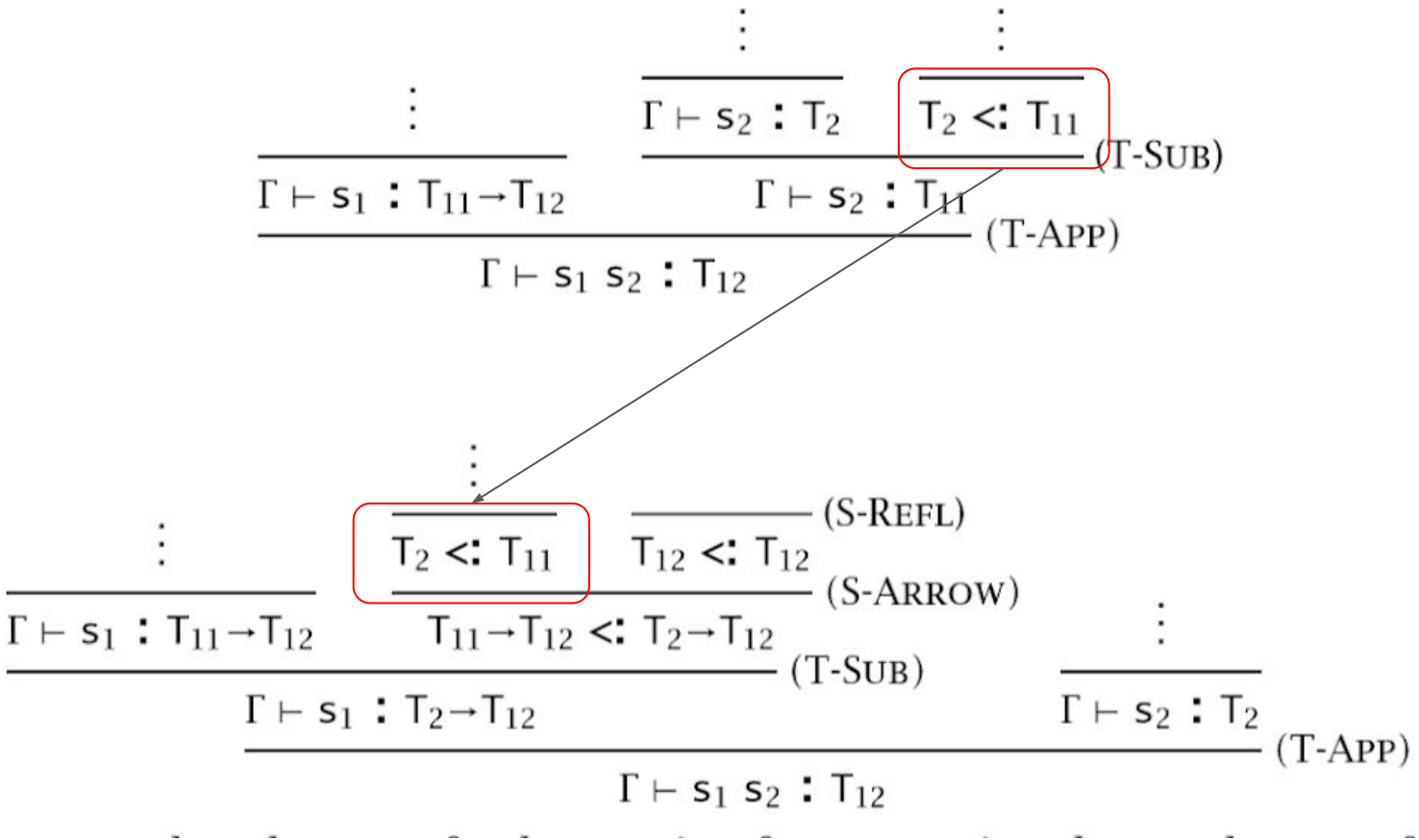
da adattare il termine ritornato passiamo ad adattare il tipo della funzione specificando meglio il tipo di ritorno



Esempio 2

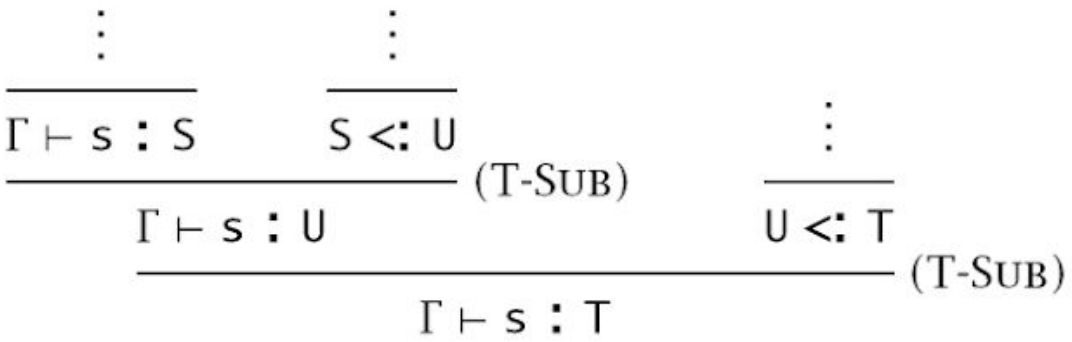

$T_{11} <: S_{11}$
$S_{12} <: T_{12}$
(S-ARROW)
$\Gamma \vdash s_1 : S_{11} \to S_{12}$
$S_{11} \to S_{12} <: T_{11} \to T_{12}$
(T-SUB)
$\Gamma \vdash s_1 : T_{11} \to T_{12}$
$\Gamma \vdash s_2 : T_{11}$
(T-APP)
$\Gamma \vdash s_1\ s_2 : T_{12}$
$\Gamma \vdash s_2 : T_{11}$
$T_{11} <: S_{11}$
(T-SUB)
$\Gamma \vdash s_1 : S_{11} \to S_{12}$
$\Gamma \vdash s_2 : S_{11}$
(T-APP)
$\Gamma \vdash s_1\ s_2 : S_{12}$
$S_{12} <: T_{12}$
(T-SUB)
$\Gamma \vdash s_1\ s_2 : T_{12}$

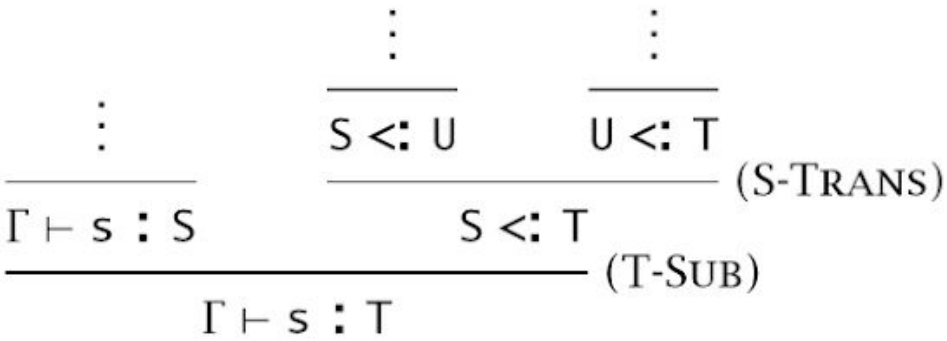
Esempio 3


$\Gamma \vdash s_1 : T_{11} \to T_{12}$
$\Gamma \vdash s_2 : T_2$
$T_2 <: T_{11}$
(T-SUB)
$\Gamma \vdash s_2 : T_{11}$
(T-APP)
$\Gamma \vdash s_1\ s_2 : T_{12}$
$T_2 <: T_{11}$
(S-REFL)
$T_{12} <: T_{12}$
(S-ARROW)
$\Gamma \vdash s_1 : T_{11} \to T_{12}$
$T_{11} \to T_{12} <: T_2 \to T_{12}$
(T-SUB)
$\Gamma \vdash s_1 : T_2 \to T_{12}$
$\Gamma \vdash s_2 : T_2$
(T-APP)
$\Gamma \vdash s_1\ s_2 : T_{12}$

Esempio 4:

collassamento tramite S-Trans





Typing Algoritmico

Attuando queste modifiche ripetutamente possiamo ottenere degli alberi in cui TSub e' usato solo in tre posti:

1. nell'albero sinistro di TApp (Necessaria, per adattare il parametro);
2. nella tipaggio della guardia di un If then else;
3. o come ultima regola dell'albero (eliminare questa non e' un problema in quanto vuol dire che possiamo tipare solo col tipo più specifico).

Typing Algoritmico

Si rimuove T-Sub

Si aggiunge T-AppAlg

La nuova regola TAppAlg ingloba TApp preceduta da TSub del parametro

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_2 \qquad T_2 <: T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}}$$

Si deve aggiungere anche T-IFAlg

La nuova regola TIfAlg ingloba TIf preceduta da TSub sulla guardia

$$\frac{\Gamma \vdash t_1 : T_1 \qquad T_1 <: \mathsf{Bool} \qquad \Gamma \vdash t_2 : T_2 \qquad \Gamma \vdash t_3 : T_3 \qquad T_2 \vee T_3 = T}{\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : T}$$

Ora siamo guidati dalla struttura del termine e possiamo tipare solo con il tipo piu' specifico. In sostanza si può scrivere l'algoritmo.

Soundness typing algoritmico

Se $\Gamma \mapsto t:T$ allora $\Gamma \vdash t:T$

Si procede per induzione sull'altezza di $\Gamma \mapsto t:T$.

Completezza typing algoritmico [tipo minimo]

Se $\Gamma \vdash t:T$ allora $\Gamma \mapsto t:S$ per qualche $S<:T$

Si procede per induzione sull'altezza di $\Gamma \vdash t:T$.

Correttezza codice

se typeOf Γ t =Just T allora $\Gamma \mapsto t:T$ [correttezza]

se $\Gamma \mapsto t:T$ allora typeOf Γ t =Just T [completezza]

se $\Gamma \mapsto t:T$ derivabile allora typeOf t =Just T altrimenti typeOf t =Nothing