CS520, Handout 25, 28 November 2006

Metatheory of Subtyping

Reading: [Pierce, Chapter 16]

Typing and subtyping rules (so far) are not syntax-directed

See [Handout 24, page 5] for:

Two different typing derivations for $(\texttt{lambda x : Real. sqRoot x})\ 5$.

Typing and subtyping rules (so far) are not syntax-directed

See [Handout 24, page 5] for:

Two different typing derivations for $(\texttt{lambda x : Real. sqRoot x})\ 5$.

Below:

Two different derivations of the relation $\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}$.

$$\dfrac{\texttt{Int} <: \texttt{Real} \qquad \texttt{Nat} <: \texttt{Int}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Arrow}$$

$$\dfrac{\dfrac{\texttt{Real} <: \texttt{Real} \qquad \texttt{Nat} <: \texttt{Int}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Real} \to \texttt{Int}}\ \textsc{S-Arrow} \qquad \dfrac{\texttt{Int} <: \texttt{Real} \qquad \texttt{Int} <: \texttt{Int}}{\texttt{Real} \to \texttt{Int} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Arrow}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Trans}$$

Typing and subtyping rules (so far) are not syntax-directed

See [Handout 24, page 5] for:

Two different typing derivations for $(\texttt{lambda x : Real. sqRoot x})\ 5$.

Below:

Two different derivations of the relation $\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}$.

$$\dfrac{\texttt{Int} <: \texttt{Real} \qquad \texttt{Nat} <: \texttt{Int}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Arrow}$$

$$\dfrac{\dfrac{\texttt{Real} <: \texttt{Real} \qquad \texttt{Nat} <: \texttt{Int}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Real} \to \texttt{Int}}\ \textsc{S-Arrow} \qquad \dfrac{\texttt{Int} <: \texttt{Real} \qquad \texttt{Int} <: \texttt{Int}}{\texttt{Real} \to \texttt{Int} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Arrow}}{\texttt{Real} \to \texttt{Nat} <: \texttt{Int} \to \texttt{Int}}\ \textsc{S-Trans}$$

How can we make the rules syntax-directed?

Restrict attention to $\lambda_{<:}$ augmented with records.

[Handout 24, pages 3 and 6] and [Pierce, Figs 15-1, 15-2, and 15-3]

Restrict attention to $\lambda_{<:}$ augmented with records.

[Handout 24, pages 3 and 6] and [Pierce, Figs 15-1, 15-2, and 15-3]

Compact version of subtyping rules [Pierce, Fig 16-1]:

$$T <: T \qquad \text{(S-REFL)}$$

$$\frac{T <: T' \qquad T' <: T''}{T <: T''} \qquad \text{(S-TRANS)}$$

$$T <: \mathtt{Top} \qquad \text{(S-TOP)}$$

$$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1' \to T_2 <: T_1 \to T_2'} \qquad \text{(S-ARROW)}$$

Restrict attention to $\lambda_{<:}$ augmented with records.

[Handout 24, pages 3 and 6] and [Pierce, Figs 15-1, 15-2, and 15-3]

Compact version of subtyping rules [Pierce, Fig 16-1]:

$$T <: T \qquad \text{(S-REFL)}$$

$$\frac{T <: T' \qquad T' <: T''}{T <: T''} \qquad \text{(S-TRANS)}$$

$$T <: \mathtt{Top} \qquad \text{(S-TOP)}$$

$$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1' \to T_2 <: T_1 \to T_2'} \qquad \text{(S-ARROW)}$$

The preceding are in [Handout 24, p 3] already. The following is new:

$$\frac{\{\ell_i{}^{i\in 1..n}\} \subseteq \{k_j{}^{j\in 1..m}\} \qquad k_j = \ell_i \text{ implies } U_j <: T_i}{\{k_j : U_j{}^{j\in 1..m}\} <: \{\ell_i : T_i{}^{i\in 1..n}\}} \qquad \text{(S-RCD)}$$

Algorithmic subtyping [Pierce, Fig 16-2, p 212]

$$T <: \mathtt{Top} \qquad \text{(SA-TOP)}$$

$$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1' \to T_2 <: T_1 \to T_2'} \qquad \text{(SA-ARROW)}$$

$$\frac{\{\ell_i{}^{i\in 1..n}\} \subseteq \{k_j{}^{j\in 1..m}\} \qquad k_j = \ell_i \text{ implies } U_j <: T_i}{\{k_j : U_j{}^{j\in 1..m}\} <: \{\ell_i : T_i{}^{i\in 1..n}\}} \qquad \text{(SA-RCD)}$$

Algorithmic subtyping [Pierce, Fig 16-2, p 212]

$$T <: \mathtt{Top} \qquad \text{(SA-TOP)}$$

$$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1' \to T_2 <: T_1 \to T_2'} \qquad \text{(SA-ARROW)}$$

$$\frac{\{\ell_i{}^{i \in 1..n}\} \subseteq \{k_j{}^{j \in 1..m}\} \qquad k_j = \ell_i \text{ implies } U_j <: T_i}{\{k_j : U_j{}^{j \in 1..m}\} <: \{\ell_i : T_i{}^{i \in 1..n}\}} \qquad \text{(SA-RCD)}$$

Proposition [Soundness and completeness of SA rules]
$T <: U$ according to the rules on this page if and only if
$T <: U$ according to the rules on page 3 of this handout.

Implementation of algorithmic subtyping [Pierce, p 212]

$\mathtt{subtype} : \mathtt{types} \times \mathtt{types} \longrightarrow \{\mathtt{true}, \mathtt{false}\}$

```
subtype(T,U) =
  IF U = Top
  THEN true
  ELSE
  IF T1 = T1 -> T2 and U = U1 -> U2
  THEN subtype(U1,T1) and subtype(T2,U2)
  ELSE
  IF T = {k_1: T_1,...,k_m: T_m} and U = {l_1: U_1,...,l_n: U_n}
  THEN {l_1,...,l_n} is a subset of {k_1,...,k_m}
       and
       for every i in {1,...,n} there is j in {1,...,m}
       such that k_j = l_i and subtype(T_j,U_i)
  ELSE false
```


Algorithmic typing [Pierce, Fig 16-3, p 217]

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(TA-VAR)}$$

$$\frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash (\lambda x : T.\, t) : T \to T'} \qquad \text{(TA-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T \to U \qquad \Gamma \vdash t_2 : T' \qquad \boxed{T' <: T}}{\Gamma \vdash t_1\, t_2 : U} \qquad \text{(TA-APP)}$$

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{\ell_1 = t_1 \ldots \ell_n = t_n\} : \{\ell_1 : T_1 \ldots \ell_n : T_n\}} \qquad \text{(TA-RCD)}$$

$$\frac{\Gamma \vdash t : \{\ell_1 : T_1 \ldots \ell_n : T_n\}}{\Gamma \vdash t.\ell_i : T_i} \qquad \text{(TA-PROJ)}$$

Soundness and completeness of algorithmic typing

[Pierce, Theorems 16.2.4 and 16.2.4, page 218]

Proposition [Soundness]
$\Gamma \vdash t : T$ according to the rules on page 5
implies
$\Gamma \vdash t : T$ according to the rules of [Pierce, Figs 15-1 and 15-2].

Proposition [Completeness]
$\Gamma \vdash t : T$ according to the rules of [Pierce, Figs 15-1 and 15-2]
implies
$\Gamma \vdash t : U$ according to the rules on page 5, for some $U <: T$.

Implementation of algorithmic typing

Not in [Pierce].

Restrict implementation to TA-Var, TA-Abs and TA-App.

$\mathtt{typeof} : \mathtt{contexts} \times \mathtt{terms} \longrightarrow \mathtt{types} \cup \{\mathtt{error}\}$

```
typeof(G,t) =
  IF t is a variable x
  THEN IF G(x) is defined THEN return G(x) ELSE ''error''
  ELSE
  IF t = t1 t2
  THEN IF typeof(G,t1) = T1 and typeof(G,t2) = T2 and
          T1 = T11 -> T12 and subtype(T2,T11)
       THEN return T12 ELSE ``error''
  ELSE
  IF t = lambda x:T. t'
  THEN IF typeof((G,x:T),t') = T'
       THEN return T -> T' ELSE ``error''
  ELSE ``error''
```


Expressions with multiple result branches

[Pierce, Section 16.3].

Is the following program phrase well-typed?

```
if true then {x=5,y=false} else {x=6,z=true}
```


Expressions with multiple result branches

[Pierce, Section 16.3].

Is the following program phrase well-typed?

`if true then {x=5,y=false} else {x=6,z=true}`

According to the rule T-If, it is not. But it is type-safe, i.e., we can safely use it as a program phrase of type $\mathtt{x{:}Nat}$.

Expressions with multiple result branches

[Pierce, Section 16.3].

Is the following program phrase well-typed?

```
if true then {x=5,y=false} else {x=6,z=true}
```

According to the rule T-If, it is not. But it is type-safe, i.e., we can safely use it as a program phrase of type $\mathtt{x{:}Nat}$.

Introduce a new rule for conditionals:

$$\frac{\Gamma \vdash t : \mathtt{Bool} \qquad \Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2 \qquad \boxed{T_1 \vee T_2 = T}}{\Gamma \vdash \mathtt{if}\ t\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : T} \quad \text{(TA-If)}$$

$T_1 \vee T_2$ is the join of T_1 and T_2, i.e., the "least common supertype" of T_1 and T_2.