Type Systems

Lecture 10 Dec. 22nd, 2004
Sebastian Maneth

http://lampwww.epfl.ch/teaching/typeSystems/2004

Today F-sub: kernel / full

1. System F-sub
2. Properties of F-sub
3. Algorithmic Typing for F-Sub
4. Algorithmic *Sub*typing for F-Sub
5. Joins and Meets

1. System F-Sub

Bounded Quantification

```
f2poly = λX<:{a:Nat}. λx:X. {orig=x, asucc=succ(x.a)};
```

Has type `∀X<:{a:Nat}. X → {orig:X, asucc:Nat}`

1. System F-Sub

Bounded Quantification

```
f2poly = λX<:{a:Nat}. λx:X. {orig=x, asucc=succ(x.a)};
```

Has type `∀X<:{a:Nat}. X → {orig:X, asucc:Nat}`

How to derive it?

```
⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
  ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

Bounded Quantification

type abstraction (remove ∀)

If bound satisfied, then term has specified type

```
X<:{a:Nat} ⊢
λx:X.{orig=x, asucc=succ(x.a)} : X → {orig:X, asucc:Nat}
-----------------------------------------------------------
   ⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
     ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

lambda abstraction (remove →)

If argument type satisfied, then result term has specified result type.

```
X<:{a:Nat},x:X ⊢
{orig=x, asucc=succ(x.a)} : {orig:X, asucc:Nat}
-----------------------------------------------------------
 X<:{a:Nat} ⊢
 λx:X.{orig=x, asucc=succ(x.a)}:X → {orig:X, asucc:Nat}
-----------------------------------------------------------
   ⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
     ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

make record (remove { })

field terms have specified types

```
X<:{a:Nat},x:X ⊢ x:X      X<:{a:Nat},x:X ⊢ succ(x.a):Nat
-----------------------------------------------------------
 X<:{a:Nat},x:X ⊢
 {orig=x, asucc=succ(x.a)} : {orig:X, asucc:Nat}
-----------------------------------------------------------
  X<:{a:Nat} ⊢
  λx:X.{orig=x, asucc=succ(x.a)}:X → {orig:X, asucc:Nat}
-----------------------------------------------------------
   ⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
     ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

what now?

```
                                X<:{a:Nat},x:X ⊢ x:{a:Nat}
                                ---------------------------
           ok                   X<:{a:Nat},x:X ⊢ x.a:Nat
                                ---------------------------
X<:{a:Nat},x:X ⊢ x:X      X<:{a:Nat},x:X ⊢ succ(x.a):Nat
-----------------------------------------------------------
 X<:{a:Nat},x:X ⊢
 {orig=x, asucc=succ(x.a)} : {orig:X, asucc:Nat}
-----------------------------------------------------------
  X<:{a:Nat} ⊢
  λx:X.{orig=x, asucc=succ(x.a)}:X → {orig:X, asucc:Nat}
-----------------------------------------------------------
   ⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
     ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

what now? → subsumption!

term may be of any subtype

```
                      ok
X<:{a:Nat},x:X ⊢ x:X      X<:{a:Nat},x:X ⊢ X<:{a:Nat}
------------------------------------------------------
           X<:{a:Nat},x:X ⊢ x:{a:Nat}
           --------------------------
    ok     X<:{a:Nat},x:X ⊢ x.a:Nat
           ---------------------------------
X<:{a:Nat},x:X ⊢ x:X    X<:{a:Nat},x:X ⊢ succ(x.a):Nat
------------------------------------------------------
X<:{a:Nat},x:X ⊢
{orig=x, asucc=succ(x.a)} : {orig:X, asucc:Nat}
------------------------------------------------------
X<:{a:Nat} ⊢
λx:X.{orig=x, asucc=succ(x.a)}:X → {orig:X, asucc:Nat}
------------------------------------------------------
⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
  ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

```
        ok                      ok
X<:{a:Nat},x:X ⊢ x:X      X<:{a:Nat},x:X ⊢ X<:{a:Nat}
------------------------------------------------------
           X<:{a:Nat},x:X ⊢ x:{a:Nat}
           --------------------------
    ok     X<:{a:Nat},x:X ⊢ x.a:Nat
           ---------------------------------
X<:{a:Nat},x:X ⊢ x:X    X<:{a:Nat},x:X ⊢ succ(x.a):Nat
------------------------------------------------------
X<:{a:Nat},x:X ⊢
{orig=x, asucc=succ(x.a)} : {orig:X, asucc:Nat}
------------------------------------------------------
X<:{a:Nat} ⊢
λx:X.{orig=x, asucc=succ(x.a)}:X → {orig:X, asucc:Nat}
------------------------------------------------------
⊢ λX<:{a:Nat}.λx:X.{orig=x, asucc=succ(x.a)}:
  ∀X<:{a:Nat}. X → {orig:X, asucc:Nat}
```

1. System F-Sub

new typing rule

type abstraction (remove ∀)

If bound satisfied, then term has specified type

$$\frac{\Gamma, X<:B \vdash t:T}{\Gamma \vdash \lambda X<:B.t \;:\; \forall X<:B.T}$$

1. System F-Sub

As in System F: to **apply** a polymorphic function,
we have to supply a concrete SUBtype C.

```
( f2poly [{a:Nat, b:Bool}] ) {a=5, b=true}
```

1. System F-Sub

As in System F: to **apply** a polymorphic function, we have to supply a concrete SUBtype C.
C<:{a:Nat}

∀X<:{a:Nat}. X → {orig:X, asucc:Nat}

f2poly [{a:Nat, b:Bool}]

{a:Nat, b:Bool} → {orig:{a:Nat,b:Bool}, asucc:Nat}

1. System F-Sub

How to derive it?

$$\frac{}{\vdash \texttt{f2poly [\{a:Nat, b:Bool\}]: \{a:Nat, b:Bool\} → \{orig:\{a:Nat,b:Bool\}, asucc:Nat\}}}$$

1. System F-Sub

How to derive it?

Applying [X/{a:Nat, b:Bool}] must give specified type

$$\frac{\vdash \texttt{f2poly: ∀X<:\{a:Nat\}.X→\{orig:X, asucc:Nat\}} \qquad \vdash \texttt{\{a:Nat, b:Bool\} <:\{a:Nat\}}}{\vdash \texttt{f2poly [\{a:Nat, b:Bool\}]: \{a:Nat, b:Bool\} → \{orig:\{a:Nat,b:Bool\}, asucc:Nat\}}}$$

1. System F-Sub

(record subtyping)

ok

ok

$$\frac{\vdash \texttt{f2poly: ∀X<:\{a:Nat\}.X→\{orig:X, asucc:Nat\}} \qquad \vdash \texttt{\{a:Nat, b:Bool\} <:\{a:Nat\}}}{\vdash \texttt{f2poly [\{a:Nat, b:Bool\}]: \{a:Nat, b:Bool\} → \{orig:\{a:Nat,b:Bool\}, asucc:Nat\}}}$$

1. System F-Sub

new typing rule

type application (remove [ ])

t polymorphic with bound X<:B C subtype of B

$$\frac{\Gamma \vdash t : \forall X{<:}B.T \qquad \Gamma \vdash C{<:}B}{\Gamma \vdash t[C] : [X/C]T}$$

1. System F-Sub

Typing Rules

→ usual lambda-rules (T-VAR, T-ABS, T-APP)

→ type abstraction

→ type application (uses subtyping on bounds!)

→ subsumption (uses subtyping!)

1. System F-Sub

Subtyping Rules

→ reflexivity, transitivity, Top (evth. is <:Top)

→ type variables: $\frac{X{<:}T \in \Gamma}{\Gamma \vdash X{<:}T}$

→ function (S-ARROW)

(covariant on result,
contravariant on argument)

1. System F-Sub

new subtyping rule (for quantified types)

$$\frac{}{\Gamma \vdash \forall X{<:}B_1.T_1 \ <: \ \forall X{<:}B_2.T_2}$$

1. System F-Sub

new subtyping rule (for quantified types)

which bound??

$$\frac{\Gamma, X<:B \ \vdash \ T_1 <: T_2}{\Gamma \ \vdash \ \forall X<:B_1.T_1 \ <: \ \forall X<:B_2.T_2}$$

1. System F-Sub

new subtyping rule (for quantified types)

Must have SAME bound B

"the *kernel* rule"

$$\frac{\Gamma, X<:B \ \vdash \ T_1 <: T_2}{\Gamma \ \vdash \ \forall X<:B.T_1 \ <: \ \forall X<:B.T_2}$$

→ with this simple rule: **Kernel F-Sub**

1. System F-Sub

"the *kernel* rule"

$$\frac{\Gamma, X<:B \ \vdash \ T_1 <: T_2}{\Gamma \ \vdash \ \forall X<:B.T_1 \ <: \ \forall X<:B.T_2}$$

`∀X<:{a:Nat}. {a:Nat,b:Bool}   <:   ∀X<:{a:Nat}. {a:Nat}`

If expected type is `∀X<:{a:Nat}. {a:Nat}`
then also fu. of type `∀X<:{a:Nat}. {a:Nat,b:Bool}` is OK!

1. System F-Sub

"the *kernel* rule"

$$\frac{\Gamma, X<:B \ \vdash \ T_1 <: T_2}{\Gamma \ \vdash \ \forall X<:B.T_1 \ <: \ \forall X<:B.T_2}$$

`∀X<:{a:Nat}. {a:Nat,b:Bool}   <:   ∀X<:{a:Nat}. {a:Nat}`

If expected type is `∀X<:{a:Nat}`. `{a:Nat}`
then also fu. of type `∀X<:{a:Nat}. {a:Nat,b:Bool}` is OK!

➔ Will only be instantiated by subtypes `X` of `{a:Nat}`

THUS, `∀X<:Top` is OK too, because it asks for less! (for the least, actually..)

`∀X<:{a:Nat,d:Nat}` NOT OK, because we only know X will be `<:{a:Nat}`

1. System F-Sub

Full F-Sub

$$\frac{\Gamma \vdash B_2 <: B_1 \qquad \Gamma, X<:B_2 \vdash T_1 <: T_2}{\Gamma \vdash \forall X<:B_1.T_1 \;<:\; \forall X<:B_2.T_2}$$

→ covariant on result

→ contravariant on bounds

2. Properties of F-Sub

Preservation

If $\Gamma \vdash t:T$ and $t \to t'$, then $\Gamma \vdash t':T$.

Progress

If t is a closed and well-typed term, then either
t is a value, or
$t \to t'$ for some term t'.

Proofs: Induction on the structure of terms.

→ Use canonical forms lemma:

If v is closed value of type $T_1 \to T_2$, then $v = \lambda x:S_1.t_2$
If v is closed value of type $\forall X<:T_1.T_2$, then $v = \lambda X<:T_1.t_2$.

3. Algorithmic Typing for F-Sub

Idea of type checking algorithm for simply typed lambda-calculus w. subtyping:

→ calculate the *minimal type* of terms

Apply same idea for F-Sub!

Now application, and then subsumtion, and all is OK!

$$\frac{\dfrac{X<:Nat\to Nat,\ y:X \;\vdash\; y\ 5:\ Nat}{X<:Nat\to Nat \;\vdash\; \lambda y:X.\ y\ 5:\ X \to Nat}}{\vdash\; \lambda X<:Nat\to Nat.\ \lambda y:X.\ y\ 5 \;:\; \forall X.X<:Nat\to Nat.\ X \to Nat}$$

3. Algorithmic Typing for F-Sub

Idea of type checking algorithm for simply typed lambda-calculus w. subtyping:

→ calculate the *minimal type* of terms

Apply same idea for F-Sub!

→ subsumtion is NOT syntax-directed!! Can we do without??

Now application, and then subsumtion, and all is OK!

$$\frac{\dfrac{X<:Nat\to Nat,\ y:X \;\vdash\; y\ 5:\ Nat}{X<:Nat\to Nat \;\vdash\; \lambda y:X.\ y\ 5:\ X \to Nat}}{\vdash\; \lambda X<:Nat\to Nat.\ \lambda y:X.\ y\ 5 \;:\; \forall X.X<:Nat\to Nat.\ X \to Nat}$$

3. Algorithmic Typing for F-Sub

→ subsumtion is NOT syntax-directed!! Can we do without??

Now application, and then subsumtion, and all is OK!

$\boxed{\mathtt{X<:Nat\to Nat,\ y:X}} \vdash \mathtt{y\ 5: Nat}$

→ y must be arrow type!

→ smallest (non-variable) arrow type, that is a supertype of X

$\Gamma \vdash \mathtt{X} \Uparrow \mathtt{Nat\to Nat}$

"X exposes to Nat→Nat under Γ"

3. Algorithmic Typing for F-Sub

Exposure:

$$\frac{\mathtt{X<:T} \in \Gamma \qquad \Gamma \vdash \mathtt{T} \Uparrow \mathtt{T'}}{\Gamma \vdash \mathtt{X} \Uparrow \mathtt{T'}} \qquad\qquad \frac{\text{T is not a type variable}}{\Gamma \vdash \mathtt{T} \Uparrow \mathtt{T}}$$

Example: $\Gamma = \mathtt{X<:Nat,\ Y<:Nat\to Nat,\ Z<:Y,\ W<:Z}$

Then $\Gamma \vdash \mathtt{Z} \Uparrow \mathtt{Nat\to Nat}$ and $\Gamma \vdash \mathtt{W} \Uparrow \mathtt{Nat\to Nat}$

Lemma. If $\Gamma \vdash S \Uparrow T$, then
(1) $\Gamma \vdash \mathtt{S<:T}$
(2) If $\Gamma \vdash \mathtt{S<:U}$ and U is not a variable, then $\Gamma \vdash \mathtt{T<:U}$.

3. Algorithmic Typing for F-Sub

New rule for application, includes argument subsumtion:

$$\frac{\begin{array}{ll}\Gamma \vdash t_1 : T_1 & \Gamma \vdash T_1 \Uparrow (D\to E) \\ \Gamma \vdash t_2 : T_2 & \Gamma \vdash T_2 <: D\end{array}}{\Gamma \vdash t_1\ t_2 : E} \text{ TA-APP}$$

In Example:

$$\frac{\begin{array}{ll}\Gamma \vdash y : X & \Gamma \vdash X \Uparrow (Nat\to Nat) \\ \Gamma \vdash 5 : Nat & \Gamma \vdash Nat <: Nat\end{array}}{\underbrace{X<:Nat\to Nat,\ y:X}_{\Gamma} \vdash y\ 5 : Nat} \text{ TA-APP}$$

3. Algorithmic Typing for F-Sub

New rule for type application, includes argument subsumtion:

Old:
$$\frac{\Gamma \vdash t : \forall X<:B.T \qquad \Gamma \vdash C<:B}{\Gamma \vdash t\ [C] : [X/C]T}$$

New:
$$\frac{\begin{array}{ll}\Gamma \vdash t : T_1 & \Gamma \vdash T_1 \Uparrow \forall X<:B.T \\ & \Gamma \vdash C<:B\end{array}}{\Gamma \vdash t\ [C] : [X/C]T} \text{ TA-TAPP}$$

3. Algorithmic Typing for F-Sub

$\vdash$ = $\underbrace{\text{T-VAR, T-ABS, T-APP}}_{\text{as in simply typed}}$, T-TABS, T-TAPP, T-SUB

$\vdash$ = T-VAR, T-ABS, TA-APP, T-TABS, TA-TAPP

Theorem. (correctness of minimal typing)

(1) If $\Gamma \vdash$ `t:T`, then $\Gamma \vdash$ `t:T` (soundness)

(2) If $\Gamma \vdash$ `t:T`, then $\Gamma \vdash$ `t:M` with $\Gamma \vdash$ `M<:T`. (completeness)

Corollary. The relation $\vdash$ is decidable, given a decision procedure for the subtype relation.

4. Algorithmic *Sub*typing for F-Sub

Subtyping Rules

```
→ reflexivity, transitivity, Top  (evth. is <:Top)
→ type variables
→ function (S-ARROW)
→ quantified types (kernel / full)
```

4. Algorithmic *Sub*typing for F-Sub

Problematic Subtyping Rules

```
→ reflexivity, transitivity, Top
→ type variables
→ function (S-ARROW)
→ quantified types (kernel / full)
```

4. Algorithmic *Sub*typing for F-Sub

Problematic Subtyping Rules

```
→ reflexivity, transitivity, Top
→ type variables
→ function (S-ARROW)
→ quantified types (kernel / full)
```

Reflexivity is ONLY needed for variables: $\Gamma \vdash$ `X<:X` not derivable without it!

→ Replace `reflexivity` by **new rule**: $\Gamma \vdash$ `X<:X`.

4. Algorithmic *Sub*typing for F-Sub

Problematic Subtyping Rules

- → Γ ⊢ X<:X, transitivity, Top
- → type variables
- → function (S-ARROW)
- → quantified types (kernel / full)

Transitivity: Γ = W<:Top,V<:W,U<:V,X<:U

Γ ⊢ Z<:W can ONLY be proved using transitivity.

$$\frac{\dfrac{X<:U \in \Gamma}{\Gamma \vdash X<:U} \qquad \dfrac{\vdots}{\Gamma \vdash U<:W}}{\Gamma \vdash X<:W}$$

4. Algorithmic *Sub*typing for F-Sub

Problematic Subtyping Rules

- → Γ ⊢ X<:X, transitivity, Top
- → type variables
- → function (S-ARROW)
- → quantified types (kernel / full)

Transitivity: Γ = T<:Top,V<:T,U<:V,X<:U

Γ ⊢ Z<:W can ONLY be proved using transitivity.

$$\frac{\dfrac{X<:U \in \Gamma}{\Gamma \vdash X<:U} \qquad \dfrac{\vdots}{\Gamma \vdash U<:T}}{\Gamma \vdash X<:T}$$

← ONLY essential use of transitivity!

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules

- → Γ ⊢ X<:X, Top
- → function (S-ARROW)
- → quantified types (kernel / full)

➔ Replace transitivity, and type vars by **new rule**:

$$\frac{X<:U \in \Gamma \qquad \Gamma \vdash U<:T}{\Gamma \vdash X<:T}$$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (kernel)

Γ ⊢ S<:Top

Γ ⊢ X<:X

$$\frac{X<:U \in \Gamma \qquad \Gamma \vdash U<:T}{\Gamma \vdash X<:T}$$

$$\frac{\Gamma \vdash T_1<:S_1 \qquad \Gamma \vdash S_2<:T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2}$$

$$\frac{\Gamma, X<:B \vdash T_1<:T_2}{\Gamma \vdash \forall X<:B.T_1 <: \forall X<:B.T_2}$$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (kernel)

$\Gamma \vdash S<:Top$

$\Gamma \vdash X<:X$

$$\frac{X<:U \in \Gamma \qquad \Gamma \vdash U<:T}{\Gamma \vdash X<:T}$$

$$\frac{\Gamma \vdash T_1<:S_1 \qquad \Gamma \vdash S_2<:T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \qquad \frac{\Gamma, X<:B \vdash T_1<:T_2}{\Gamma \vdash \forall X<:B.T_1 <: \forall X<:B.T_2}$$

Theorem. $\Gamma \vdash S<:T$ if and only if $\Gamma \vdash S<:T$.

Theorem. The subtyping algorithm terminates on all inputs.

→ Subtyping in kernel F-sub is decidable.

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL)

$\Gamma \vdash S<:Top$

$\Gamma \vdash X<:X$

$$\frac{X<:U \in \Gamma \qquad \Gamma \vdash U<:T}{\Gamma \vdash X<:T}$$

$$\frac{\Gamma \vdash T_1<:S_1 \qquad \Gamma \vdash S_2<:T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \qquad \xcancel{\frac{\Gamma, X<:B \vdash T_1<:T_2}{\Gamma \vdash \forall X<:B.T_1 <: \forall X<:B.T_2}}$$

$$\frac{\Gamma \vdash B_2<:B_1 \qquad \Gamma, X<:B_2 \vdash T_1<:T_2}{\Gamma \vdash \forall X<:B_1.T_1 <: \forall X<:B_2.T_2}$$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL) **do not terminate!!**
→Construct a vicious circle:

Define $\neg S = \forall X<:S.X$

Then, $\Gamma \vdash \neg S <: \neg T$ iff $T<:S$. (Contravariance on bounds..)

$T := \forall X<:Top.\ \neg(\forall Y<:X.\ \neg Y)$

Try to show that

$X_0<:T \vdash X_0 <: \forall X_1<:X_0.\neg X_1$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL) **do not terminate!!**
→Construct a vicious circle:

Define $\neg S = \forall X<:S.X$

Then, $\Gamma \vdash \neg S <: \neg T$ iff $T<:S$. (Contravariance on bounds..)

$T := \forall X<:Top.\ \neg(\forall Y<:X.\ \neg Y)$

Try to show that

$X_0<:T \vdash X_0 <: \forall X_1<:X_0.\neg X_1$
$X_0<:T \vdash \forall X_1<:Top.\neg(\forall X_2<:X_1.\neg X_2) <: \forall X_1<:X_0.\neg X_1$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL) **do not terminate!!**
→Construct a vicious circle:

Define $\neg S = \forall X<:S.X$

Then, $\Gamma \vdash \neg S <: \neg T$ iff $T<:S$. (Contravariance on bounds..)

$T := \forall X<:Top.\ \neg(\forall Y<:X.\ \neg Y)$

Try to show that

$$X_0<:T \vdash X_0 <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T \vdash \forall X_1<:Top.\neg(\forall X_2<:X_1.\neg X_2) <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T, X_1<:X_0 \vdash \neg(\forall X_2<:X_1.\neg X_2) <: \neg X_1$$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL) **do not terminate!!**
→Construct a vicious circle:

Define $\neg S = \forall X<:S.X$

Then, $\Gamma \vdash \neg S <: \neg T$ iff $T<:S$. (Contravariance on bounds..)

$T := \forall X<:Top.\ \neg(\forall Y<:X.\ \neg Y)$

Try to show that

$$X_0<:T \vdash X_0 <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T \vdash \forall X_1<:Top.\neg(\forall X_2<:X_1.\neg X_2) <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T, X_1<:X_0 \vdash \neg(\forall X_2<:X_1.\neg X_2) <: \neg X_1$$
$$X_0<:T, X_1<:X_0 \vdash X_1 <: \forall X_2<:X_1.\neg X_2.$$

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (FULL) **do not terminate!!**
→Construct a vicious circle:

Define $\neg S = \forall X<:S.X$

Then, $\Gamma \vdash \neg S <: \neg T$ iff $T<:S$. (Contravariance on bounds..)

$T := \forall X<:Top.\ \neg(\forall Y<:X.\ \neg Y)$

Try to show that

$$X_0<:T \vdash X_0 <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T \vdash \forall X_1<:Top.\neg(\forall X_2<:X_1.\neg X_2) <: \forall X_1<:X_0.\neg X_1$$
$$X_0<:T, X_1<:X_0 \vdash \neg(\forall X_2<:X_1.\neg X_2) <: \neg X_1$$
$$X_0<:T, X_1<:X_0 \vdash X_1 <: \forall X_2<:X_1.\neg X_2.$$

5. Joins and Meets

How to type if-then-else, in the presence of subsumtion?

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

What is the type of this term?

→ `{x:Bool}` — take the *least* (most precise) common supertype of S and T

<:

or `{x:Top}`? = "the *join* of S and T"

<:

or `{}`? =: $S \vee T$

<:

or `Top`?

$$\frac{t_1 : Bool \quad t_2 : T_2 \quad t_3 : T_3 \quad T = T_2 \vee T_3}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$$

5. Joins and Meets

$\Gamma \vdash S \vee T :=$

$\Gamma \vdash S \vee T :=$	T	if $\Gamma \vdash S<:T$
	S	if $\Gamma \vdash T<:S$
	J	if $S=X$, $X<:U \in \Gamma$, and $\Gamma \vdash U \vee T=J$
	J	if $T=X$, $X<:U \in \Gamma$, and $\Gamma \vdash S \vee U=J$
	$M \to J$	if $S=S_1 \to S_2$, $T=T_1 \to T_2$, $\Gamma \vdash S_1 \wedge T_1=M$, and $\Gamma \vdash S_1 \vee T_2=J$
	$\forall X<:U.J_2$	if $S= \forall X<:U.S_2$, $T= \forall X<:U.T_2$, $\Gamma, X<:U \vdash S_2 \vee T_2=J_2$
	Top	otherwise

5. Joins and Meets

In kernel F-sub:

→ every pair S,T has (effectively) a join

→ if S and T have at least one subtype in common, then they have (effectively) a meet

In full F-sub: NO / NO

Summary

formalism	comput. power	type checking	type reconstr.
Simply typed lambda calculus	normalizing	lin.time	poly.time
Simply typed lambda calculus + subtyping	normalizing	lin.time	poly.time
Featherweight Java	r.e. compl.	lin.time	
Let-Polymorphism / Prenex-Polymorphism	normalizing	lin.time	EXPTIME
System F	normalizing	lin.time	UNDEC.
System F + subtyping (kernel)	normalizing	poly.time	UNDEC.
System F + subtyping (full)	normalizing	UNDEC.	UNDEC.