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 $\forall$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

$$\frac{\texttt{X<:\{a:Nat\}} \vdash \lambda\texttt{x:X.\{orig=x, asucc=succ(x.a)\}} : \texttt{X} \rightarrow \texttt{\{orig:X, asucc:Nat\}}}{\vdash \lambda\texttt{X<:\{a:Nat\}.}\lambda\texttt{x:X.\{orig=x, asucc=succ(x.a)\}:} \;\forall\texttt{X<:\{a:Nat\}. X} \rightarrow \texttt{\{orig:X, asucc:Nat\}}}$$

1. System F-Sub

lambda abstraction (remove →)

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

$$\frac{\dfrac{X<:\{a:Nat\},x:X \vdash \{orig=x,\ asucc=succ(x.a)\} : \{orig:X,\ asucc:Nat\}}{X<:\{a:Nat\} \vdash \lambda x:X.\{orig=x,\ asucc=succ(x.a)\}:X \to \{orig:X,\ asucc:Nat\}}}{\vdash \lambda X<:\{a:Nat\}.\lambda x:X.\{orig=x,\ asucc=succ(x.a)\}: \forall X<:\{a:Nat\}.\ X \to \{orig:X,\ asucc:Nat\}}$$

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

```
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?

```
⊢ 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

⊢ f2poly: ∀X<:{a:Nat}.X→{orig:X, asucc:Nat}   ⊢ {a:Nat, b:Bool} <:{a:Nat}

⊢ f2poly [{a:Nat, b:Bool}]:
{a:Nat, b:Bool} → {orig:{a:Nat,b:Bool}, asucc:Nat}

1. System F-Sub

(record subtyping)

ok

ok

$\vdash$ f2poly: ∀X<:{a:Nat}.X→{orig:X, asucc:Nat}

$\vdash$ {a:Nat, b:Bool} <:{a:Nat}

$\vdash$ 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 $t \vdash t:T$ 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!

$$X <: Nat \to Nat,\ y : X \ \vdash\ y\ 5 : Nat$$

$$X <: Nat \to Nat \ \vdash\ \lambda y : X.\ y\ 5 : X \to Nat$$

$$\vdash\ \lambda X <: Nat \to Nat.\ \lambda y : X.\ y\ 5 \ :\ \forall X.X <: Nat \to Nat.\ X \to Nat$$

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!

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

3. Algorithmic Typing for F-Sub

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

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

`X<:Nat→Nat, y:X` ⊢ `y 5: Nat`

→ y must be arrow type!

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

$\Gamma \vdash$ `X` $\Uparrow$ `Nat→Nat`

"`X exposes to Nat→Nat under` Γ"

3. Algorithmic Typing for F-Sub

Exposure:

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

Example: Γ = X<:Nat, Y<:Nat→Nat, Z<:Y, W<:Z

Then $\Gamma \vdash$ Z $\Uparrow$ Nat→Nat and $\Gamma \vdash$ W $\Uparrow$ Nat→Nat

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

3. Algorithmic Typing for F-Sub

New rule for application, includes argument subsumtion:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash T_1 \Uparrow (D \rightarrow E) \qquad \Gamma \vdash t_2 : T_2 \qquad \Gamma \vdash T_2 <: D}{\Gamma \vdash t_1\ t_2 : E} \quad \text{TA-APP}$$

In Example:

$$\frac{\Gamma \vdash y : X \qquad \Gamma \vdash X \Uparrow (Nat \rightarrow Nat) \qquad \Gamma \vdash 5 : Nat \qquad \Gamma \vdash Nat <: Nat}{\underbrace{X <: Nat \rightarrow Nat,\ y : X}_{\Gamma} \vdash y\ 5 : Nat} \quad \text{TA-APP}$$

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:

$$\frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 \Uparrow \forall X <: B.T \qquad \Gamma \vdash C <: B}{\Gamma \vdash t\ [C] : [X/C]T} \quad \text{TA-TAPP}$$

3. Algorithmic Typing for F-Sub

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

(T-VAR, T-ABS, T-APP: as in simply typed)

$\vdash$ = 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

→ $\Gamma \vdash$ `X<:X`, `transitivity`, `Top`
→ `type variables`
→ `function (S-ARROW)`
→ `quantified types (kernel / full)`

Transitivity: $\Gamma =$ `W<:Top,V<:W,U<:V,X<:U`

$\Gamma \vdash$ `Z<:W` can ONLY be proved using transitivity.

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

4. Algorithmic *Sub*typing for F-Sub

Problematic Subtyping Rules

- → $\Gamma \vdash$ `X<:X`, transitivity, `Top`
- → type variables
- → function (S-ARROW)
- → quantified types (kernel / full)

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

$\Gamma \vdash$ `Z<:W` can ONLY be proved using transitivity.

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

← ONLY essential use of transitivity!

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules

- → $\Gamma \vdash$ `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)

$\Gamma \vdash S <: Top$

$\Gamma \vdash X <: X$

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

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

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

4. Algorithmic *Sub*typing for F-Sub

Algorithmic Subtyping Rules (kernel)

$$\Gamma \;{\color{red}\vdash}\; \mathtt{S<:Top}$$

$$\Gamma \;{\color{red}\vdash}\; \mathtt{X<:X}$$

$$\frac{\mathtt{X<:U} \in \Gamma \qquad \Gamma \;{\color{red}\vdash}\; \mathtt{U<:T}}{\Gamma \;{\color{red}\vdash}\; \mathtt{X<:T}}$$

$$\frac{\Gamma \;{\color{red}\vdash}\; \mathtt{T_1<:S_1} \qquad \Gamma \;{\color{red}\vdash}\; \mathtt{S_2<:T_2}}{\Gamma \;{\color{red}\vdash}\; \mathtt{S_1 \to S_2 <: T_1 \to T_2}}$$

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

Theorem. $\Gamma \;{\color{red}\vdash}\; \mathtt{S<:T}$ if and only if $\Gamma \vdash \mathtt{S<:T}$.

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)

$$\Gamma \vdash S <: Top$$

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

$$\Gamma \vdash X <: X$$

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

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

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

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

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

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

$$
\begin{array}{llcl}
X_0{<:}T & \vdash X_0 & <: & \forall X_1{<:}X_0.\neg X_1 \\
X_0{<:}T & \vdash \forall X_1{<:}Top.\neg(\forall X_2{<:}X_1.\neg X_2) & <: & \forall X_1{<:}X_0.\neg X_1 \\
X_0{<:}T, X_1{<:}X_0 & \vdash \neg(\forall X_2{<:}X_1.\neg X_2) & <: & \neg X_1 \\
X_0{<:}T, X_1{<:}X_0 & \vdash X_1 & <: & \forall X_2{<:}X_1.\neg X_2.
\end{array}
$$

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

$$
\begin{array}{llll}
X_0 <: T & \vdash & X_0 & <: \forall X_1 <: X_0.\neg X_1 \\
X_0 <: T & \vdash \forall X_1 <: Top.\neg(\forall X_2 <: X_1.\neg X_2) & & <: \forall X_1 <: X_0.\neg X_1 \\
X_0 <: T, X_1 <: X_0 & \vdash \neg(\forall X_2 <: X_1.\neg X_2) & & <: \neg X_1 \\
X_0 <: T, X_1 <: X_0 & \vdash & X_1 & <: \forall X_2 <: X_1.\neg X_2.
\end{array}
$$

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}`

<:

or `{x:Top}`?

<:

or `{}`?

<:

or `Top`?

take the *least* (most precise) common supertype of S and T

= "the *join* of S and T"

=: $S \vee T$

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

5. Joins and Meets

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

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 $\Gamma \vdash S \vee U = J$
$M \rightarrow J$	if	$S = S_1 \rightarrow S_2$, $T = T_1 \rightarrow T_2$, $\Gamma \vdash S_1 \wedge T_1 = M$, 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.