CIS 500

Software Foundations

Fall 2002

30 October

Administrivia

- Prof. Pierce out of town Nov. 5 – 14
  - No office hours Nov 5, 7, 12, or 14
  - Next Wednesday: guest lecturer (on Chapter 16)
  - Following Monday: review session (led by Anne and Jim)
  - 3PM recitation cancelled on Nov 11 – go to Max's in Towne 307 instead
  - Following Wednesday: Midterm II
- There will be class on the Wednesday before Thanksgiving (Nov. 27)

Review

Subtyping

Intuitions: `S <: T` means...

- "An element of `S` may safely be used wherever an element of `T` is expected." (Official.)
- `S` is "better than" `T`.
- `S` is a subset of `T`.
- `S` is more informative / richer than `T`.

Subtype relation

$$\mathtt{S} <: \mathtt{S} \qquad \text{(S-Refl)}$$

$$\frac{\mathtt{S} <: \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}} \qquad \text{(S-Trans)}$$

$$\{\mathtt{l}_i{:}\mathtt{T}_i \ ^{i\in 1..n+k}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i \ ^{i\in 1..n}\} \qquad \text{(S-RcdWidth)}$$

$$\frac{\text{for each } i \qquad \mathtt{S}_i <: \mathtt{T}_i}{\{\mathtt{l}_i{:}\mathtt{S}_i \ ^{i\in 1..n}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i \ ^{i\in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

$$\frac{\{\mathtt{k}_j{:}\mathtt{S}_j \ ^{j\in 1..n}\} \text{ is a permutation of } \{\mathtt{l}_i{:}\mathtt{T}_i \ ^{i\in 1..n}\}}{\{\mathtt{k}_j{:}\mathtt{S}_j \ ^{j\in 1..n}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i \ ^{i\in 1..n}\}} \qquad \text{(S-RcdPerm)}$$

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \qquad \mathtt{S}_2 <: \mathtt{T}_2}{\mathtt{S}_1 \to \mathtt{S}_2 <: \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(S-Arrow)}$$

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

Subsumption Rule

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{S} \qquad \mathtt{S} <: \mathtt{T}}{\Gamma \vdash \mathtt{t} : \mathtt{T}} \qquad \text{(T-Sub)}$$

Properties

Safety

Statements of progress and preservation theorems are unchanged.

Proofs become a bit more involved, because the typing relation is no longer syntax directed.

Preservation

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

Proof: By induction on typing derivations.

(Which cases are hard?)

Subsumption case

Case T-SUB: $t : S \quad S <: T$

By the induction hypothesis, $\Gamma \vdash t' : S$. By T-SUB, $\Gamma \vdash t : T$.

Subsumption case

Case T-SUB: $t : S \quad S <: T$

By the induction hypothesis, $\Gamma \vdash t' : S$. By T-SUB, $\Gamma \vdash t : T$.

Not hard!

Application case

Case T-App:

$t = t_1\ t_2 \qquad \Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11} \qquad T = T_{12}$

From the evaluation rules (i.e., strictly speaking, from the inversion lemma for evaluation), there are three rules by which $t \longrightarrow t'$ can be derived: E-App1, E-App2, and E-AppAbs. Proceed by cases.

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

Application case

Case T-App:

$t = t_1\ t_2 \qquad \Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11} \qquad T = T_{12}$

From the evaluation rules (i.e., strictly speaking, from the inversion lemma for evaluation), there are three rules by which $t \longrightarrow t'$ can be derived: E-App1, E-App2, and E-AppAbs. Proceed by cases.

Subcase E-App1: $t_1 \longrightarrow t_1' \qquad t' = t_1'\ t_2$

The result follows from the induction hypothesis and T-App.

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-App1)}$$

Case T-App (continued):

$t = t_1\ t_2 \qquad \Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11} \qquad T = T_{12}$

Subcase E-App2: $t_1 = v_1 \qquad t_2 \longrightarrow t_2' \qquad t' = v_1\ t_2'$

Similar.

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-App2)}$$

Case T-App (continued):

$t = t_1\ t_2 \qquad \Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11} \qquad T = T_{12}$

Subcase E-AppAbs: $t_1 = \lambda x{:}S_{11}.\ t_{12} \qquad t_2 = v_2 \qquad t' = [x \mapsto v_2]t_{12}$

By the inversion lemma for the typing relation...

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \qquad \text{(E-AppAbs)}$$

Case T-App (continued):

$t = t_1\ t_2 \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11} \quad T = T_{12}$

Subcase E-AppAbs: $t_1 = \lambda x{:}S_{11}.\ t_{12} \quad t_2 = v_2 \quad t' = [x \mapsto v_2]t_{12}$

By the inversion lemma for the typing relation... $T_{11} <: S_{11}$ and $\Gamma, x{:}S_{11} \vdash t_{12} : T_{12}$.

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Case T-App (continued):

$t = t_1\ t_2 \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11} \quad T = T_{12}$

Subcase E-AppAbs: $t_1 = \lambda x{:}S_{11}.\ t_{12} \quad t_2 = v_2 \quad t' = [x \mapsto v_2]t_{12}$

By the inversion lemma for the typing relation... $T_{11} <: S_{11}$ and $\Gamma, x{:}S_{11} \vdash t_{12} : T_{12}$.

By T-Sub, $\Gamma \vdash t_2 : S_{11}$.

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Case T-App (continued):

$t = t_1\ t_2 \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11} \quad T = T_{12}$

Subcase E-AppAbs: $t_1 = \lambda x{:}S_{11}.\ t_{12} \quad t_2 = v_2 \quad t' = [x \mapsto v_2]t_{12}$

By the inversion lemma for the typing relation... $T_{11} <: S_{11}$ and $\Gamma, x{:}S_{11} \vdash t_{12} : T_{12}$.

By T-Sub, $\Gamma \vdash t_2 : S_{11}$.

By the substitution lemma, $\Gamma \vdash t' : T_{12}$, and we are done.

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type).

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type). Need another lemma...

> Lemma: If $U <: T_1 \to T_2$, then U has the form $U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. (Proof: by induction on subtyping derivations.)

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type). Need another lemma...

> Lemma: If $U <: T_1 \to T_2$, then U has the form $U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$.

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type). Need another lemma...

> Lemma: If $U <: T_1 \to T_2$, then U has the form $U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$.

The IH now applies, yielding $U_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : U_2$.

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type). Need another lemma...

> Lemma: If $U <: T_1 \to T_2$, then U has the form $U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$.

The IH now applies, yielding $U_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : U_2$.

From $U_1 <: S_1$ and $T_1 <: U_1$, rule S-Trans gives $T_1 <: S_1$.

Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x{:}S_1.s_2 : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda x{:}S_1.s_2 : U \quad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that S is an arrow type). Need another lemma...

> Lemma: If $U <: T_1 \to T_2$, then U has the form $U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \to U_2$, with $T_1 <: U_1$ and $U_2 <: T_2$.

The IH now applies, yielding $U_1 <: S_1$ and $\Gamma, x{:}S_1 \vdash s_2 : U_2$.

From $U_1 <: S_1$ and $T_1 <: U_1$, rule S-Trans gives $T_1 <: S_1$.

From $\Gamma, x{:}S_1 \vdash s_2 : U_2$ and $U_2 <: T_2$, rule T-Sub gives $\Gamma, x{:}S_1 \vdash s_2 : T_2$, and we are done.

Subtyping and Other Features

Ascription and Casting

Ordinary ascription:

$$\frac{\Gamma \vdash t_1 : T}{\Gamma \vdash t_1 \text{ as } T : T} \quad \text{(T-Ascribe)}$$

$$v_1 \text{ as } T \longrightarrow v_1 \quad \text{(E-Ascribe)}$$

Ascription and Casting

Ordinary ascription:

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-Ascribe)}$$

$$\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1} \qquad \text{(E-Ascribe)}$$

Casting (cf. Java):

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{S}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-Cast)}$$

$$\frac{\vdash \mathtt{v_1} : \mathtt{T}}{\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1}} \qquad \text{(E-Cast)}$$

Subtyping and Variants

$$\mathtt{<l_i{:}T_i}^{\ i\in 1..n}\mathtt{>} \ <: \ \mathtt{<l_i{:}T_i}^{\ i\in 1..n+k}\mathtt{>} \qquad \text{(S-VariantWidth)}$$

$$\frac{\text{for each } i \quad \mathtt{S_i} <: \mathtt{T_i}}{\mathtt{<l_i{:}S_i}^{\ i\in 1..n}\mathtt{>} \ <: \ \mathtt{<l_i{:}T_i}^{\ i\in 1..n}\mathtt{>}} \qquad \text{(S-VariantDepth)}$$

$$\frac{\mathtt{<k_j{:}S_j}^{\ j\in 1..n}\mathtt{>} \text{ is a permutation of } \mathtt{<l_i{:}T_i}^{\ i\in 1..n}\mathtt{>}}{\mathtt{<k_j{:}S_j}^{\ j\in 1..n}\mathtt{>} \ <: \ \mathtt{<l_i{:}T_i}^{\ i\in 1..n}\mathtt{>}} \qquad \text{(S-VariantPerm)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1}}{\Gamma \vdash \mathtt{<l_1=t_1>} : \mathtt{<l_1{:}T_1>}} \qquad \text{(T-Variant)}$$

Subtyping and Lists

$$\frac{\mathtt{S_1} <: \mathtt{T_1}}{\mathtt{List\ S_1} <: \mathtt{List\ T_1}} \qquad \text{(S-List)}$$

I.e., `List` is a covariant type constructor.

Subtyping and References

$$\frac{\mathtt{S_1} <: \mathtt{T_1} \qquad \mathtt{T_1} <: \mathtt{S_1}}{\mathtt{Ref\ S_1} <: \mathtt{Ref\ T_1}} \qquad \text{(S-Ref)}$$

I.e., `Ref` is *not* a covariant (nor a contravariant) type constructor.

Subtyping and Arrays

Similarly...

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-Array)}$$

Subtyping and Arrays

Similarly...

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-Array)}$$

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-ArrayJava)}$$

This is regarded (even by the Java designers) as a mistake in the design.

References again

Observation: a value of type `Ref T` can be used in two different ways: as a source for values of type `T` and as a sink for values of type `T`.

References again

Observation: a value of type `Ref T` can be used in two different ways: as a source for values of type `T` and as a sink for values of type `T`.

Idea: Split `Ref T` into three parts:

- `Source T`: reference cell with “read cabability”
- `Sink T`: reference cell with “write cabability”
- `Ref T`: cell with both capabilities

Modified Typing Rules

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Source}\ \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{!t}_1 : \mathtt{T}_{11}} \quad \text{(T-Deref)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Sink}\ \mathtt{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1 \mathtt{:=} \mathtt{t}_2 : \mathtt{Unit}} \quad \text{(T-Assign)}$$

Subtyping rules

$$\frac{\mathtt{S}_1 <: \mathtt{T}_1}{\mathtt{Source}\ \mathtt{S}_1 <: \mathtt{Source}\ \mathtt{T}_1} \quad \text{(S-Source)}$$

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1}{\mathtt{Sink}\ \mathtt{S}_1 <: \mathtt{Sink}\ \mathtt{T}_1} \quad \text{(S-Sink)}$$

$$\mathtt{Ref}\ \mathtt{T}_1 <: \mathtt{Source}\ \mathtt{T}_1 \quad \text{(S-RefSource)}$$

$$\mathtt{Ref}\ \mathtt{T}_1 <: \mathtt{Sink}\ \mathtt{T}_1 \quad \text{(S-RefSink)}$$

Capabilities

Other kinds of capabilities (e.g., send and receive capabilities on communication channels, encrypt/decrypt capabilities of cryptographic keys, ...) can be treated similarly.

Coercion semantics

[skip]

Intersection Types

The inhabitants of $T_1 \wedge T_2$ are terms belonging to both S and T—i.e., $T_1 \wedge T_2$ is an order-theoretic meet (greatest lower bound) of T_1 and T_2.

$$T_1 \wedge T_2 <: T_1 \qquad \text{(S-INTER1)}$$

$$T_1 \wedge T_2 <: T_2 \qquad \text{(S-INTER2)}$$

$$\frac{S <: T_1 \qquad S <: T_2}{S <: T_1 \wedge T_2} \qquad \text{(S-INTER3)}$$

$$S \to T_1 \wedge S \to T_2 <: S \to (T_1 \wedge T_2) \qquad \text{(S-INTER4)}$$

Intersection Types

Intersection types permit a very flexible form of finitary overloading.

$$+ : (\mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Nat}) \wedge (\mathtt{Float} \to \mathtt{Float} \to \mathtt{Float})$$

This form of overloading is extremely powerful.

Every strongly normalizing untyped lambda-term can be typed in the simply typed lambda-calculus with intersection types.

$\longrightarrow$ type reconstruction problem is undecidable

Intersection types have not been used much in language designs (too powerful!), but are being intensively investigated as type systems for intermediate languages in highly optimizing compilers (cf. Church project).

Union types

Union types are also useful.

$T_1 \vee T_2$ is an untagged (non-disjoint) union of T_1 and T_2

$\longrightarrow$ no `case` construct. The only operations we can safely perform on elements of $T_1 \backslash/ T2$ are ones that make sense for both T_1 and T_2.

N.b.: untagged `union` types in C are a source of type safety violations precisely because they ignores this restriction, allowing any operation on an element of $T_1 \vee T_2$ that makes sense for either T_1 or T_2.

Union types are being used recently in type systems for XML processing languages (cf. XDuce, Xtatic).