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)

Subtyping

Intuitions: $S \ll T$ 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

- \( S \subseteq S \) \hspace{1cm} (S-REFL)
- \( S \subseteq U \hspace{0.5cm} U \subseteq T \hspace{1cm} (S-TRANS) \)
- \( S \subseteq T \)

\[
\begin{align*}
\{l_i : T_i \mid i \in l_i.\text{set} \} &\subseteq \{l_i : T_i \mid i \in l_i.\text{set} \} \\
\text{for each } i &\hspace{1cm} S_i \subseteq T_i \\
\{l_i : T_i \mid i \in l_i.\text{set} \} &\subseteq \{l_i : T_i \mid i \in l_i.\text{set} \} \\
\{k_i : S_i \mid i \in l_i.\text{set} \} &\text{is a permutation of } \{l_i : T_i \mid i \in l_i.\text{set} \}
\end{align*}
\]

\( \{k_i : S_i \mid i \in l_i.\text{set} \} \subseteq \{l_i : T_i \mid i \in l_i.\text{set} \} \) \hspace{1cm} (S-RCDPERM)

### Subsumption Rule

\[
\Gamma \vdash t : S \hspace{1cm} S \subseteq T \\
\hspace{1cm} \Gamma \vdash t : T \\
\]

\( \Gamma \vdash t : T \) \hspace{1cm} (T-SUB)
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 $t \rightarrow t'$, then $\Gamma \vdash t' : T$.
Proof: By induction on typing derivations.
(Which cases are hard?)

Subsumption case

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

Subsumption case

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

Not hard!
From the evaluation rules (i.e., strictly speaking, from the inversion lemma for evaluation), there are three rules by which $t \rightarrow t'$ can be derived: E-APP1, E-APP2, and E-APPABS. Proceed by cases.

\[
\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \\
\frac{}{\Gamma \vdash t_1 \cdot t_2 : T_2} \quad \text{(T-APP)}
\]

Similar.

By the inversion lemma for the typing relation...

\[
\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \\
\frac{}{\Gamma \vdash \lambda x : T_1 . \, t_2 : T_1 \rightarrow T_2} \\
\frac{}{(\lambda x : T_1 . \, t_2) \, \cdot v_2 : [x \mapsto v_2] T_1 T_2} \quad \text{(E-APPABS)}
\]
Case T-APP (CONTINUED):
\[ t = t_1 t_2 \quad \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \quad T = T_2 \]

Subcase E-APPABS:
\[ t_1 = \lambda x : S_1: . \quad t_1 t_2 = t_2 \quad \Gamma' = [x \mapsto t_2] t_2 \]
By the inversion lemma for the typing relation... \( T_1 \ll S_1 \) and \( \Gamma, x : S_1 \vdash t_2 : T_2 \).

\[ \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \quad T = T_2 \]

\[ \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \quad T = T_2 \]

Subcase E-APPABS:
\[ t_1 = \lambda x : S_1: . \quad t_2 = t_2 \quad \Gamma' = [x \mapsto t_2] t_2 \]
By the inversion lemma for the typing relation... \( T_1 \ll S_1 \) and \( \Gamma, x : S_1 \vdash t_2 : T_2 \).

By T-SUB, \( \Gamma \vdash t_2 : S_1 \).

\[ \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \quad T = T_2 \]

\[ \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \quad T = T_2 \]

Inversion Lemma

**Lemma:** If \( \Gamma \vdash \lambda x : S_1 : t_2 : T_1 \rightarrow T_2 \), then \( T_1 \ll S_1 \) and \( \Gamma, x : S_1 \vdash t_2 : T_2 \).

**Proof:** Induction on typing derivations.
Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x : S_1 . s_2 : T_1 \rightarrow T_2$, then $T_1 \preceq S_1$ and $\Gamma, x : S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-SUB: $\lambda x : S_1 . s_2 : U \quad U \preceq T_1 \rightarrow T_2$

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 $U \preceq T_1 \rightarrow T_2$, then $U$ has the form $U_1 \rightarrow U_2$, with $T_1 \preceq U_1$ and $U_2 \preceq T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \rightarrow U_2$, with $T_1 \preceq U_1$ and $U_2 \preceq T_2$. The IH now applies, yielding $U_1 \preceq S_1$ and $\Gamma, x : S_1 \vdash s_2 : U_2$. 

CIS 500, 30 October 15-a
Inversion Lemma

Lemma: If $\Gamma \vdash \lambda x : S_1 . s_2 : T_1 \rightarrow T_2$, then $T_1 \preceq S_1$ and $\Gamma, x : S_1 \vdash s_2 : T_2$.

Proof: Induction on typing derivations.

Case T-SUB: $\lambda x : S_1 . s_2 : U \quad U \preceq T_1 \rightarrow T_2$

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 $U \preceq T_1 \rightarrow T_2$, then $U$ has the form $U_1 \rightarrow U_2$, with $T_1 \preceq U_1$ and $U_2 \preceq T_2$. (Proof: by induction on subtyping derivations.)

By this lemma, we know $U = U_1 \rightarrow U_2$, with $T_1 \preceq U_1$ and $U_2 \preceq T_2$. The IH now applies, yielding $U_1 \preceq S_1$ and $\Gamma, x : S_1 \vdash s_2 : U_2$.

From $U_1 \preceq S_1$ and $T_1 \preceq U_1$, rule S-TRANS gives $T_1 \preceq S_1$.

Ascription and Casting

Ordinary ascription:

\[ \Gamma \vdash t_1 : T \]
\[ \Gamma \vdash t_1 \text{ as } T : T \quad (T-ASCRIBE) \]
\[ v_1 \text{ as } T \rightarrow v_1 \quad (E-ASCRIBE) \]
Ascription and Casting

Ordinary ascription:

\[ \Gamma \vdash t_1 : T \]

\[ \Gamma \vdash t_1 \text{ as } T : T \]

\[ v_1 \text{ as } T \rightarrow v_1 \]

(T-ASCRIBE)

(Casting (cf. Java):

\[ \Gamma \vdash t_1 : S \]

\[ \Gamma \vdash t_1 \text{ as } T : T \]

\[ \Gamma \vdash v_1 : T \]

\[ v_1 \text{ as } T \rightarrow v_1 \]

(T-CAST)

\[ \text{for each } i, S_i \preceq T_i \]

\[ \langle l_i : S_i \rangle \preceq \langle l_i : T_i \rangle \]

(E-CAST)

Subtyping and Variants

\[ \langle l_i : T_i \rangle \preceq \langle l_i : T_i \rangle \]

(S-VARIANTWIDTH)

\[ \langle l_i : S_i \rangle \preceq \langle l_i : T_i \rangle \]

(S-VARIANTDEPTH)

\[ \langle l_i : S_i \rangle \preceq \langle l_i : T_i \rangle \]

(S-VARIANTPERM)

\[ \Gamma \vdash t_1 : T_1 \]

\[ \Gamma \vdash \langle l_i = t_i \rangle : \langle l_i : T_i \rangle \]

(T-VARIANT)

Subtyping and Lists

\[ S_i \preceq T_1 \]

List \[ S_i \preceq \text{List} T_1 \]

(S-LIST)

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

Subtyping and References

\[ S_i \preceq T_1 \]

\[ T_1 \preceq S_i \]

Ref \[ S_i \preceq \text{Ref} T_1 \]

(S-REF)

I.e., Ref is not a covariant (nor a contravariant) type constructor.
Subtyping and Arrays

Similarly...

\[
\begin{align*}
S_1 & \subseteq T_1 & T_1 & \subseteq S_1 \\
\text{Array } S_1 & \subseteq \text{Array } T_1 & (S\text{-ARRAY})
\end{align*}
\]

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`.

Idea: Split `Ref T` into three parts:

- **Source T**: reference cell with “read capability”
- **Sink T**: reference cell with “write capability”
- **Ref T**: cell with both capabilities
Modified Typing Rules

\[
\begin{align*}
& \Gamma, \Sigma \vdash t_1 : \text{Source } T_{11} \\
& \quad \Gamma, \Sigma \vdash !t_1 : T_{11} \quad \text{(T-DEREF)} \\
& \Gamma, \Sigma \vdash t_1 : \text{Sink } T_{11} \quad \Gamma, \Sigma \vdash t_2 : T_{11} \\
& \quad \Gamma, \Sigma \vdash t_1 := t_2 : \text{Unit} \quad \text{(T-ASSIGN)}
\end{align*}
\]

Subtyping rules

\[
\begin{align*}
& S_1 \ll T_1 \\
& \quad \text{Source } S_1 \ll \text{Source } T_1 \quad \text{(S-SOURCE)} \\
& T_1 \ll S_1 \\
& \quad \text{Sink } S_1 \ll \text{Sink } T_1 \quad \text{(S-SINK)} \\
& \text{Ref } T_1 \ll \text{Source } T_1 \\
& \quad \text{Ref } T_1 \ll \text{Sink } T_1 \quad \text{(S-REFSOURCE)} \\
& \quad \text{Ref } T_1 \ll \text{Sink } T_1 \quad \text{(S-REFSINK)}
\end{align*}
\]

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

\[
T_1 \land T_2 \leq T_1 \quad \text{(S-INTER1)} \\
T_1 \land T_2 \leq T_2 \quad \text{(S-INTER2)} \\
S \leq T_1 \quad S \leq T_2 \\
\overline{S \leq T_1 \land T_2} \quad \text{(S-INTER3)} \\
S \rightarrow T_1 \land S \rightarrow T_2 \leq S \rightarrow (T_1 \land T_2) \quad \text{(S-INTER4)}
\]

Intersection types permit a very flexible form of finitary overloading.

\[+ : (\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}) \land (\text{Float} \rightarrow \text{Float} \rightarrow \text{Float})\]

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.

\[\rightarrow \text{ type reconstruction problem is undecidable}\]

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.

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

\[-\text{no case construct. The only operations we can safely perform on elements of } T_1 \lor T_2 \text{ are ones that make sense for both } T_1 \text{ and } T_2.\]

N.b.: untagged union types in C are a source of type safety violations precisely because they ignore this restriction, allowing any operation on an element of $T_1 \lor T_2$ 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).