SNU 4541.574
Programming Language Theory

ack: from BCP's slides

Subtyping

Motivation

With our usual typing rule for applications

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

the term

$$\mathtt{(\lambda r{:}\{x{:}Nat\}.\ r.x)\ \{x{=}0,y{=}1\}}$$

is *not* well typed.

Motivation

With our usual typing rule for applications

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

the term

$$\mathtt{(}\lambda\mathtt{r:\{x:Nat\}.\ r.x)\ \{x=0,y=1\}}$$

is *not* well typed.

But this is silly: all we're doing is passing the function a *better* argument than it needs.

Polymorphism

A *polymorphic* function may be applied to many different types of data.

Varieties of polymorphism:

- Parametric polymorphism (ML-style)
- Subtype polymorphism (OO-style)
- Ad-hoc polymorphism (overloading)

Our topic for the next few lectures is *subtype* polymorphism, which is based on the idea of *subsumption*.

Subsumption

More generally: some *types* are better than others, in the sense that a value of one can always safely be used where a value of the other is expected.

We can formalize this intuition by introducing

1. a *subtyping* relation between types, written $\mathtt{S} <: \mathtt{T}$
2. a rule of *subsumption* stating that, if $\mathtt{S} <: \mathtt{T}$, then any value of type $\mathtt{S}$ can also be regarded as having type $\mathtt{T}$

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

Example

We will define subtyping between record types so that, for example,

$$\texttt{\{x:Nat, y:Nat\}} <: \texttt{\{x:Nat\}}$$

So, by subsumption,

$$\vdash \texttt{\{x=0,y=1\} : \{x:Nat\}}$$

and hence

$$\texttt{(}\lambda\texttt{r:\{x:Nat\}. r.x) \{x=0,y=1\}}$$

is well typed.

The Subtype Relation: Records

"Width subtyping" (forgetting fields on the right):

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

Intuition: `{x:Nat}` is the type of all records with *at least* a numeric `x` field.

Note that the record type with *more* fields is a *sub*type of the record type with fewer fields.

Reason: the type with more fields places a *stronger constraint* on values, so it describes *fewer values*.

The Subtype Relation: Records

Permutation of fields:

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

By using S-RcdPerm together with S-RcdWidth and S-Trans allows us to drop arbitrary fields within records.

The Subtype Relation: Records

"Depth subtyping" within fields:

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

The types of individual fields may change.

Example

$$\dfrac{\dfrac{}{\texttt{\{a:Nat,b:Nat\}} <: \texttt{\{a:Nat\}}}\ \textsc{S-RcdWidth} \qquad \dfrac{}{\texttt{\{m:Nat\}} <: \texttt{\{\}}}\ \textsc{S-RcdWidth}}{\texttt{\{x:\{a:Nat,b:Nat\},y:\{m:Nat\}\}} <: \texttt{\{x:\{a:Nat\},y:\{\}\}}}\ \textsc{S-RcdDepth}$$

Variations

Real languages often choose not to adopt all of these record subtyping rules. For example, in Java,

- A subclass may not change the argument or result types of a method of its superclass (i.e., no depth subtyping)
- Each class has just one super*class* ("single inheritance" of classes)

  $\longrightarrow$ *each class member (field or method) can be assigned a single index, adding new indices "on the right" as more members are added in subclasses (i.e., no permutation for classes)*

- A class may implement multiple *interfaces* ("multiple inheritance" of interfaces)
  I.e., permutation is allowed for interfaces.

The Subtype Relation: Arrow types

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \qquad \text{(S-Arrow)}$$

Note the order of T_1 and S_1 in the first premise. The subtype relation is *contravariant* in the left-hand sides of arrows and *covariant* in the right-hand sides.

Intuition: if we have a function f of type $S_1 \to S_2$, then we know that f accepts elements of type S_1; clearly, f will also accept elements of any subtype T_1 of S_1. The type of f also tells us that it returns elements of type S_2; we can also view these results belonging to any supertype T_2 of S_2. That is, any function f of type $S_1 \to S_2$ can also be viewed as having type $T_1 \to T_2$.

The Subtype Relation: Top

It is convenient to have a type that is a supertype of every type. We introduce a new type constant `Top`, plus a rule that makes `Top` a maximum element of the subtype relation.

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

Cf. `Object` in Java.

The Subtype Relation: General rules

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

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

Subtype relation

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

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

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

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

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

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

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

Properties of Subtyping

Safety

Statements of progress and preservation theorems are unchanged from $\lambda_{\to}$.

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

Given a derivation, we don't always know what rule was used in the last step. The rule T-SUB could appear anywhere.

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

Preservation

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

Proof: By induction on typing derivations.

(Which cases are likely to be hard?)

Subsumption case

Case T-SUB: `t : S` `S <: T`

Subsumption case

Case T-Sub: $\quad \mathtt{t} : \mathtt{S} \qquad \mathtt{S} <: \mathtt{T}$

By the induction hypothesis, $\Gamma \vdash \mathtt{t}' : \mathtt{S}$. By T-Sub, $\Gamma \vdash \mathtt{t} : \mathtt{T}$.

Subsumption case

Case T-SUB: $\quad \mathtt{t : S} \qquad \mathtt{S <: T}$

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

Not hard!

Application case

Case T-APP:

$$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} {\to} \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$$

By the inversion lemma for evaluation, there are three rules by which $\mathtt{t} \longrightarrow \mathtt{t}'$ can be derived: E-APP1, E-APP2, and E-APPABS. Proceed by cases.

Application case

Case T-App:

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}{\to}\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three rules by which $\mathtt{t} \longrightarrow \mathtt{t}'$ can be derived: E-App1, E-App2, and E-AppAbs. Proceed by cases.

Subcase E-App1: $\qquad \mathtt{t}_1 \longrightarrow \mathtt{t}_1' \qquad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}{\to}\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

Application case

Case T-APP:

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three rules by which $\mathtt{t} \longrightarrow \mathtt{t}'$ can be derived: E-APP1, E-APP2, and E-APPABS. Proceed by cases.

Subcase E-APP1: $\qquad \mathtt{t}_1 \longrightarrow \mathtt{t}_1' \qquad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(E-APP1)}$$

Case T-App (continued):

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}{\rightarrow}\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

Subcase E-App2: $\quad \mathtt{t}_1 = \mathtt{v}_1 \qquad \mathtt{t}_2 \longrightarrow \mathtt{t}_2' \qquad \mathtt{t}' = \mathtt{v}_1\ \mathtt{t}_2'$

Similar.

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}{\rightarrow}\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(E-App2)}$$

Case T-App (continued):

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

Subcase E-AppAbs:

$\mathtt{t}_1 = \lambda \mathtt{x} {:} \mathtt{S}_{11}.\ \mathtt{t}_{12} \qquad \mathtt{t}_2 = \mathtt{v}_2 \qquad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

By the inversion lemma for the typing relation...

Case T-App (continued):

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}{\rightarrow}\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

Subcase E-AppAbs:

$\mathtt{t}_1 = \lambda\mathtt{x}{:}\mathtt{S}_{11}.\ \mathtt{t}_{12} \qquad \mathtt{t}_2 = \mathtt{v}_2 \qquad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

By the inversion lemma for the typing relation... $\mathtt{T}_{11}$ <: $\mathtt{S}_{11}$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_{11} \vdash \mathtt{t}_{12} : \mathtt{T}_{12}$.

Case T-App (continued):

$$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}\to\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$$

Subcase E-AppAbs:

$$\mathtt{t}_1 = \lambda\mathtt{x}{:}\mathtt{S}_{11}.\ \mathtt{t}_{12} \qquad \mathtt{t}_2 = \mathtt{v}_2 \qquad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$$

By the inversion lemma for the typing relation... $\mathtt{T}_{11}$ <: $\mathtt{S}_{11}$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_{11} \vdash \mathtt{t}_{12} : \mathtt{T}_{12}$.
By T-Sub, $\Gamma \vdash \mathtt{t}_2 : \mathtt{S}_{11}$.

Case T-App (continued):

$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11} \qquad \mathtt{T} = \mathtt{T}_{12}$

Subcase E-AppAbs:

$\mathtt{t}_1 = \lambda\mathtt{x}{:}\mathtt{S}_{11}.\ \mathtt{t}_{12} \qquad \mathtt{t}_2 = \mathtt{v}_2 \qquad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

By the inversion lemma for the typing relation... $\mathtt{T}_{11} <: \mathtt{S}_{11}$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_{11} \vdash \mathtt{t}_{12} : \mathtt{T}_{12}$.
By T-Sub, $\Gamma \vdash \mathtt{t}_2 : \mathtt{S}_{11}$.
By the substitution lemma, $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$, and we are done.

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

$$(\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-AppAbs)}$$

Inversion Lemma for Typing

Lemma: If $\Gamma \vdash \lambda\mathtt{x}:\mathtt{S}_1.\mathtt{s}_2 : \mathtt{T}_1 \rightarrow \mathtt{T}_2$, then $\mathtt{T}_1 <: \mathtt{S}_1$ and $\Gamma, \mathtt{x}:\mathtt{S}_1 \vdash \mathtt{s}_2 : \mathtt{T}_2$.

Proof: Induction on typing derivations.

Inversion Lemma for Typing

Lemma: If $\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{S}_1\mathtt{.s}_2 : \mathtt{T}_1 \to \mathtt{T}_2$, then $\mathtt{T}_1$ <: $\mathtt{S}_1$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_1 \vdash \mathtt{s}_2 : \mathtt{T}_2$.

Proof: Induction on typing derivations.

Case T-Sub: $\lambda \mathtt{x}{:}\mathtt{S}_1\mathtt{.s}_2 : \mathtt{U}$ $\qquad$ $\mathtt{U}$ <: $\mathtt{T}_1 \to \mathtt{T}_2$

Inversion Lemma for Typing

Lemma: If $\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{S}_1\mathtt{.s}_2 : \mathtt{T}_1 {\rightarrow} \mathtt{T}_2$, then $\mathtt{T}_1$ <: $\mathtt{S}_1$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_1 \vdash \mathtt{s}_2 : \mathtt{T}_2$.

Proof: Induction on typing derivations.

Case T-SUB: $\quad \lambda \mathtt{x}{:}\mathtt{S}_1\mathtt{.s}_2 : \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}_1 {\rightarrow} \mathtt{T}_2$

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

Inversion Lemma for Typing

Lemma: If $\Gamma \vdash \lambda\mathtt{x}{:}\mathtt{S}_1\,.\,\mathtt{s}_2 : \mathtt{T}_1 \to \mathtt{T}_2$, then $\mathtt{T}_1$ <: $\mathtt{S}_1$ and $\Gamma, \mathtt{x}{:}\mathtt{S}_1 \vdash \mathtt{s}_2 : \mathtt{T}_2$.

Proof: Induction on typing derivations.

Case T-SUB: $\lambda\mathtt{x}{:}\mathtt{S}_1\,.\,\mathtt{s}_2 : \mathtt{U}$ $\qquad \mathtt{U}$ <: $\mathtt{T}_1 \to \mathtt{T}_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that $\mathtt{U}$ 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 for Typing

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: $\lambda x{:}S_1.s_2 : U \qquad U$ <: $T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that U 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 for Typing

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: $\lambda x{:}S_1.s_2 : U \qquad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that U 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 for Typing

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: $\lambda x{:}S_1.s_2 : U \qquad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that U 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 for Typing

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: $\lambda x{:}S_1.s_2 : U \qquad U <: T_1 \to T_2$

We want to say "By the induction hypothesis...", but the IH does not apply (we do not know that U 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 with Other Features

Ascription and Casting

Ordinary ascription:

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

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

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

$$\texttt{<}\texttt{l}_i\texttt{:}\texttt{T}_i\ ^{i\in 1..n}\texttt{>} \quad \texttt{<:} \quad \texttt{<}\texttt{l}_i\texttt{:}\texttt{T}_i\ ^{i\in 1..n+k}\texttt{>} \qquad \text{(S-VariantWidth)}$$

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

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1}{\Gamma \vdash \texttt{<}\texttt{l}_1\texttt{=}\texttt{t}_1\texttt{>} : \texttt{<}\texttt{l}_1\texttt{:}\texttt{T}_1\texttt{>}} \qquad \text{(T-Variant)}$$

Subtyping and Lists

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

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

Subtyping and References

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

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

Subtyping and References

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

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

- When a reference is *read*, the context expects a $\mathtt{T_1}$, so if $\mathtt{S_1 <: T_1}$ then an $\mathtt{S_1}$ is ok.

Subtyping and References

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

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

- When a reference is *read*, the context expects a $\texttt{T}_1$, so if $\texttt{S}_1 \texttt{ <: } \texttt{T}_1$ then an $\texttt{S}_1$ is ok.
- When a reference is *written*, the context provides a $\texttt{T}_1$ and if the actual type of the reference is $\texttt{Ref S}_1$, someone else may use the $\texttt{T}_1$ as an $\texttt{S}_1$. So we need $\texttt{T}_1 \texttt{ <: } \texttt{S}_1$.

Subtyping and Arrays

Similarly...

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

Subtyping and Arrays

Similarly...

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

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

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

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

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

$$\mathtt{Ref\ T_1} <: \mathtt{Source\ T_1} \qquad \text{(S-RefSource)}$$

$$\mathtt{Ref\ T_1} <: \mathtt{Sink\ T_1} \qquad \text{(S-RefSink)}$$