Subtyping:

\[
\{ x = 0 \} \cdot x 
\]

\[
\{ y = 1 \} \cdot y 
\]

These are not well typed:

\[
\{ x = 0 \} \cdot x \times 1 
\]

\[
\{ y = 1 \} \cdot y 
\]

With our usual typing rule for applications:

\[
\{ x = 0 \} \cdot x \times 1 
\]

\[
\{ y = 1 \} \cdot y 
\]

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)

In this class, we will consider subtype polymorphism, which is based on the idea of subsumption.

In programming, some types are better than others, in the sense that a value of one type can be used where a value of the other is expected.

Example

\[ \{ x : \text{Nat} \} \]

can be regarded as having type \( T \) if \( S \rightarrow T \). Then any value of type \( S \) can also be regarded as having type \( T \).

Subsumption

We can formalize this intuition by introducing a rule of subsumption between types, written \( S \rightarrow T \).

\[ \frac{\alpha : S \rightarrow T}{\alpha : T} \]

We can also regard this rule as introducing a subtype relation between types, written \( S \rightarrow T \).

\[ \frac{S \rightarrow T}{S <: T} \]

Example

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

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

So, by subsumption,

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

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

\[ \{ x : \text{Nat} \} \]

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

we can form the intuition by introducing a rule of subsumption between types, written \( S \rightarrow T \).

\[ \frac{\alpha : S \rightarrow T}{\alpha : T} \]

We can also regard this rule as introducing a subtype relation between types, written \( S \rightarrow T \).

\[ \frac{S \rightarrow T}{S <: T} \]

Example

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

\[ \{ x : \text{Nat} \} \]

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

So, by subsumption,

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

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

\[ \{ x : \text{Nat} \} \]

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]

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

\[ \{ x : \text{Nat} \} \]

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

So, by subsumption,

\[ \{ x : \text{Nat}, y : \text{Nat} \} \]

\[ \{ x : \text{Nat} \} \]
The Subtype Relation: Records

\[ \text{Width subtyping (forgetting fields on the right):} \]
\[ \{ l: T \}_{i \geq 1} \vdash \{ l: T \}_{i \geq 1} + k \rightarrow \{ l: T \}_{i \geq 1} \]

Intuition: \( \{ x: \text{Nat} \} \) is the type of all records with at least one numeric field.

Notethat therecord typewith more fields isa subtype of therecord typewith fewer fields.

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

CIS500, 14 November 7

The Subtype Relation: Records

\[ \text{Permutation of fields:} \]
\[ \{ k: S \}_{j \geq 1} \vdash \{ l: T \}_{i \geq 1} \vdash \{ k: S \}_{j \geq 1} \]

By using S-RcdPerm together with S-RcdWidth and S-Trans, we can drop arbitrary fields within records.

\[ \text{Depth subtyping within fields:} \]
\[ \text{foreach } i \in S \rightarrow \{ l: T \}_{i \geq 1} \rightarrow \{ l: T \}_{i \geq 1} \]

The types of individual fields may change.

Example

\[ \text{S-RcdWidth} \]
\[ \{ a: \text{Nat}, b: \text{Nat} \} \vdash \{ a: \text{Nat} \} \]
\[ \{ m: \text{Nat} \} \vdash \{ \} \]

\[ \text{S-RcdDepth} \]
\[ \{ x: \{ a: \text{Nat}, b: \text{Nat} \}, y: \{ m: \text{Nat} \} \} \vdash \{ x: \{ a: \text{Nat} \}, y: \{ \} \} \]

CIS500, 14 November 10

\[ \text{S-RcdPerm} \]
\[ \{ l: T \}_{i \geq 1} \vdash \{ l: T \}_{i \geq 1} \vdash \{ l: T \}_{i \geq 1} \]

The Subtype Relation: Records

\[ \text{Depth subtyping within fields:} \]
\[ \text{foreach } i \in S \rightarrow \{ l: T \}_{i \geq 1} \rightarrow \{ l: T \}_{i \geq 1} \]

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

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

Initial: \( \{ x: \text{Nat} \} \) in the type of all records with at least a numeric field.

\[ \text{S-RcdPerm} \]
\[ \{ l: T \}_{i \geq 1} \vdash \{ l: T \}_{i \geq 1} \vdash \{ l: T \}_{i \geq 1} \]

The Subtype Relation: Records

\[ \text{Depth subtyping within fields:} \]
\[ \text{foreach } i \in S \rightarrow \{ l: T \}_{i \geq 1} \rightarrow \{ l: T \}_{i \geq 1} \]
Variations

Reallanguagesoftenchoosentoadoptnotalloftheserecordsubtypingrules. Forexample,inJava,
Asubclassmaynotchangetheargumentorresulttypesofamethodofits superclass(i.e.,nodepthsubtyping)
eachclasshasjustonesuperclass(\singleinheritance"ofclasses)
eachclassmember(fieldormethod)canbeassignedasingle index,addingnewindices"asmoremembersare addedinsubclasses(i.e.,nopermutationforclasses)
Aclassmayimplementmultiple interfaces(\multipleinheritance"ofinterfaces)
I.e.,permutationisallowedforinterfaces.

TheSubtypeRelation:Arrowtypes

\( T_1 \triangleleft S_1 \triangleleft S_2 \triangleleft T_2 \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)

\( S \triangleleft S \triangleleft T \)
Properties of Subtyping

Safety

\[
\begin{align*}
(T\text{-Ref}) & \quad \frac{T \vdash t : T}{s : S} \\
\quad & \quad \frac{T \vdash \top : T}{T \vdash t : T} \\
\quad & \quad \frac{T \vdash s : S}{T \vdash s : S}
\end{align*}
\]

The rule \(T\text{-Ref}\) could appear anywhere. Given a derivation, we don't always know which rule was used in the last step.

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

Safety of progress and preservation theorems are unchanged from \(\lambda\).
Preservation Theorem:
If \( t : T \) and \( t \mapsto t' \), then \( t' : T \).

Proof:
By induction on typing derivations.

Subsumption case
Case T-Sub:
\( t : S \preceq T \).
By the induction hypothesis, \( t : S \preceq S \). By T-Sub, \( t : T \).

(Which cases are hard?)
By the substitution lemma,

\[ T - \text{Sub} \]

and

\[ S <: T \]

1. Case $T - \text{App}$ (continued):

- $\exists x : T. [(x : T) \Rightarrow (\text{E-App1} (0 = t \cdot t) \cdot t)]$
- $\exists x : S. [(x : S) \Rightarrow (\text{E-App2} (2 = t \cdot t) \cdot t)]$

The result follows from the induction hypotheses and $T - \text{App}$.

Subcase $E - \text{App1}$: $\exists x : T. [(x : T) \Rightarrow (\text{E-App1} (0 = t \cdot t) \cdot t)]$

can be derived. $E - \text{App2}$, $E - \text{App1}$, and $E - \text{App2}$, Proceed by cases.

The inversion lemma for evaluation, there are three rules by which $t \cdot t$

$T - \text{App}(\text{continued})$.

Similar:

Subcase $E - \text{App2}$: $\exists x : S. [(x : S) \Rightarrow (\text{E-App2} (2 = t \cdot t) \cdot t)]$

can be derived. $E - \text{App1}$, $E - \text{App2}$, and $E - \text{App1}$, Proceed by cases.

By the inversion lemma for evaluation, there are three rules by which $t \cdot t$

$T - \text{App}(\text{continued})$. Application case

Application case
By the inversion lemma for the typing relation... $T_1 < S_1$ and

\[
\Gamma \vdash \alpha : S_1, \quad \Gamma \vdash \beta : T_1
\]

Case T-App (continued):

\[
t_1 : t_1 \vdash t_2 : T_2, \quad \Gamma \vdash t_1 : T_1, \quad T - T_2
\]
We are done.

By the IH hypothesis we know \( u \rightarrow u' \) with \( T \rightarrow u \) and \( u \rightarrow u' \.

\[
\begin{align*}
\text{Lemma: } & u \rightarrow u' \text{ then } u \text{ has the form } u' = u_1 \cdots u_n, \text{ with } T \rightarrow u_1 \text{ and } u' = u_2 \cdots u_n. \\
\text{Proof:} & \text{by induction on subtyping derivations.}
\end{align*}
\]

We want to say \"by the induction hypothesis,\" but the IH does not apply.

\[
\begin{align*}
\text{Case T-Sub: } & x : S \rightarrow \gamma . \text{ then } T \rightarrow u \text{ and } u \rightarrow u'. \\
\text{Proof:} & \text{by induction on \( u' \).}
\end{align*}
\]

We do not know \( u' \) is an arrow type. Need another lemma...

\[
\begin{align*}
\text{Case T-Sub: } & x : S \rightarrow \gamma . \text{ then } T \rightarrow u \text{ and } u \rightarrow u'. \\
\text{Proof:} & \text{by induction on \( u' \).}
\end{align*}
\]

\[
\begin{align*}
\text{Case T-Sub: } & x : S \rightarrow \gamma . \text{ then } T \rightarrow u \text{ and } u \rightarrow u'. \\
\text{Proof:} & \text{by induction on \( u' \).}
\end{align*}
\]

The IH now applies, yielding \( u \rightarrow u' \) with \( T \rightarrow u \) and \( u \rightarrow u' \.

\[
\begin{align*}
\text{Case T-Sub: } & x : S \rightarrow \gamma . \text{ then } T \rightarrow u \text{ and } u \rightarrow u'. \\
\text{Proof:} & \text{by induction on \( u' \).}
\end{align*}
\]
Inversion Lemma for Typing

Lemma:
If \( x : S \rightarrow T \) and \( T \rightarrow U \), then \( T \rightarrow U \) and \( x : S \rightarrow T \).

Proof:
Induction on typing derivations.

Case T-Sub:
\[ x : S \rightarrow T \text{ and } T \rightarrow U \]
\[ u : T \rightarrow U \]
\[ \text{Sub} \]

We want to say \( \text{By the induction hypothesis...} \)
But the IH does not apply.

By this lemma, we know \( U = U_1 \rightarrow U_2 \), with \( T_1 \rightarrow U_1 \) and \( U_2 \rightarrow T_2 \).

The IH now applies, yielding \( \text{By the induction hypothesis...} \)
But the IH does not apply.

From \( u : T \rightarrow U \) and \( T \rightarrow U \), rule \( \text{S-Trans} \) gives \( T \rightarrow U \).

From \( u : T \rightarrow U \) and \( T \rightarrow U \), rule \( \text{T-Sub} \) gives \( u : T \rightarrow U \), and we are done.

Ordinary ascription:

Ordinary ascription:
\[ \text{T-Ascribe} \]
\[ \text{E-Ascribe} \]

Casting (cf. Java):

\[ \text{T-Cast} \]
\[ \text{E-Cast} \]
Ascription and Casting

Ordinary ascription:

\[ t_1 : T \quad \Rightarrow \quad T(t_1) \]

E-Ascribe

\[ v_1 : T \quad \Rightarrow \quad E(v_1) \]

Casting (cf. Java):

\[ t_1 : S \quad \Rightarrow \quad T(t_1) \]

T-Cast

\[ v_1 : T \quad \Rightarrow \quad E(v_1) \]

E-Cast

Subtyping and Variants

\[ T_1 \quad :: \quad T_2 \quad \Rightarrow \quad \langle T_1 \quad \quad T_2 \rangle \]

S-Ref

\[ S_1 \quad :: \quad T_1 \quad \Rightarrow \quad \langle S_1 \quad T_1 \rangle \]

T-Ref

\[ S_1 \quad :: \quad T_1 \quad \Rightarrow \quad \langle S_1 \quad T_1 \rangle \]

T-List

\[ S_1 \quad :: \quad T_1 \quad \Rightarrow \quad \langle S_1 \quad T_1 \rangle \]

S-List

Subtyping and Lists

Casting (cf. Java):

\[ T \quad \Rightarrow \quad \langle T \rangle \]

E-Ascribe

\[ T \quad \Rightarrow \quad \langle T \rangle \]

T-Ascribe

Ordinary ascription:

\[ T \quad \Rightarrow \quad \langle T \rangle \]
Subtyping and References

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

\[
\text{Ref} \times \text{Ref} \Rightarrow \text{Ref} \\
S_1 \Rightarrow T_1 \\
S_1 \Rightarrow T_1
\]

\[
\text{Ref} \times \text{Ref} \Rightarrow \text{Ref} \\
S_1 \Rightarrow T_1 \\
S_1 \Rightarrow T_1
\]

Similarly...

Similarly...

Subtyping and Arrays

Similarly...

Similarly...

Subtyping and Arrays

Subtyping and References

Subtyping and References
Observation: A value of type RefT 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 RefT into three parts:
- SourceT: reference cell with read capability
- SinkT: reference cell with write capability
- RefT: cell with both capabilities

Subtyping Rules

\begin{align*}
\text{(S-Source)} & \quad \text{RefT} \triangleq \text{SourceT} \\
\text{(S-RefSource)} & \quad \text{RefT} \triangleq \text{SourceT} \\
\text{(S-RefSink)} & \quad \text{SourceT} \triangleq \text{SinkT} \\
\text{(S-Sink)} & \quad \text{SourceT} \triangleq \text{SinkT} \\
\text{(S-Deref)} & \quad \text{SourceT} \triangleq \text{SinkT}
\end{align*}
Syntax-directed sets of rules

Syntax-directed rules

Technically, the reason this works is that we can divide the "positions" of the typing relation into input positions and output positions. From the subexpressions of inputs to the main goal (if any) appear in the premises (so we can calculate outputs from the main goal) and in the conclusions (so we can calculate inputs to the "subgoals" appearing in the conclusion) we can calculate inputs into the "input positions" (1) and output positions (T).
Non-syntax-directedness of typing

1. When we extend the system with subtyping, both aspects of syntax-directedness get broken.

   1. The set of typing rules now includes two rules that can be used to prove a given typing.
   
   [\[ t : S \Rightarrow T \]
   
   \[ t : T \Rightarrow S \]
   
   Non-syntax-directedness of typing

   2. Moreover, the subtyping relation is not syntax-directed either.

   : T-Sub:

   \[ t : S \Rightarrow T \]
   
   \[ t : T \Rightarrow S \]

   Non-syntax-directedness of subtyping

Non-syntax-directedness of subtyping

Moreover, the subtyping relation is not syntax-directed either.

1. There are lots of ways to derive a given subtyping statement.

   [\[ L: t : S \Rightarrow T \]
   
   \[ L: t : S \Rightarrow T \]

   [\[ L: t : S \Rightarrow T \]
   
   Whatcho do?

   1. Observation: We don’t need 1000 ways to prove a given typing or subtyping statement.

   2. Use the resulting intuition to formulatenew “algorithmic” (i.e., syntax-directed) typing and subtyping relations.

   3. Provethatthe algorithmic relations are the same as “the original ones in an appropriate sense.”

1. To implement the rule for verifying the “algorithmic relation,” we’d have to guess a value for U.

   [\[ S \Rightarrow T \]
   
   Where we can get rid of excess flexibility — think more carefully about the typing and subtyping systems to see

   subtyping statements — one at a time.

   1. Observation: We don’t need 1000 waystoprove a given typing or subtyping statement.

   2. Use the resulting intuition to formulatenew “algorithmic” (i.e., syntax-directed) typing and subtyping relations.

   3. Provethatthe algorithmic relations are the same as “the original ones in an appropriate sense.”