MTAT.05.105 Type Theory

Subtyping

Records

- Types: $\tau ::= \ldots \mid \{l_i : \tau_i{}^{i \in 1..n}\}$ record type
- Terms: $e ::= \ldots \mid \{l_i = e_i{}^{i \in 1..n}\}$ record $\mid e.l$ projection
- Values: $v ::= \ldots \mid \{l_i = v_i{}^{i \in 1..n}\}$ record value
- Example:

$$\{x = \mathtt{true},\ y = 0\} \quad : \quad \{x : \mathrm{Bool},\ y : \mathrm{Nat}\}$$

Evaluation rules for records

$$\{l_i = v_i \,^{i \in 1..n}\}.l_j \quad \rightarrow \quad v_j \qquad \text{(E-ProjRcd)}$$

$$\frac{e \rightarrow e'}{e.l \rightarrow e'.l} \qquad \text{(E-Proj)}$$

$$\frac{e_j \rightarrow e'_j}{\begin{array}{c}\{l_i = v_i \,^{i \in 1..j-1},\ l_j = e_j,\ l_k = e_k \,^{k \in j+1..n}\} \\ \rightarrow \quad \{l_i = v_i \,^{i \in 1..j-1},\ l_j = e'_j,\ l_k = e_k \,^{k \in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

Typing rules for records

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \cdots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \{l_i = e_i{}^{i \in 1..n}\} \;:\; \{l_i : \tau_i{}^{i \in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash e : \{l_i : \tau_i{}^{i \in 1..n}\}}{\Gamma \vdash e.l_j : \tau_j} \qquad \text{(T-Proj)}$$

Records: example

- Consider a term:

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

Records: example

- Consider a term:

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

- Evaluation:

$$\begin{aligned}
&(\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 1 - 1,\ y = 0 + 1\}\\
&\to (\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 0 + 1\}\\
&\to (\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\}\\
&\to \{x = 0,\ y = 1\}.x\\
&\to 0
\end{aligned}$$

Records: example

- Consider a term:

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

- Evaluation:

$$\begin{aligned}
&(\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 1 - 1,\ y = 0 + 1\}\\
&\to (\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 0 + 1\}\\
&\to (\lambda r : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\}\\
&\to \{x = 0,\ y = 1\}.x\\
&\to 0
\end{aligned}$$

- Typing derivation:

$$\dfrac{\dfrac{\dfrac{}{r : R \vdash r : R}}{r : R \vdash r.x : \mathrm{Nat}}}{\vdash (\lambda r : R.\ r.x) : R \to \mathrm{Nat}} \qquad \dfrac{\dfrac{\vdots}{\vdash 1 - 1 : \mathrm{Nat}} \qquad \dfrac{\vdots}{\vdash 0 + 1 : \mathrm{Nat}}}{\vdash \{x = 1 - 1,\ y = 0 + 1\} : R}$$

$$\vdash (\lambda r : R.\ r.x)\ \{x = 1 - 1,\ y = 0 + 1\} : \mathrm{Nat}$$

where $R \equiv \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}$.

Subtyping: motivation

- In the current system (ie. simply typed λ-calculus with records) the following term is not well-typed:

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

  - But clearly, it behaves well.

Subtyping: motivation

- In the current system (ie. simply typed λ-calculus with records) the following term is not well-typed:

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

  - But clearly, it behaves well.

- The subtyping relation tries to capture the principle of safe substitution:
  - Sometimes terms of type τ can always safely be used where terms of type σ are expected.
  - We say that τ is a subtype of σ, and write this as $\tau <: \sigma$.

Subtyping: subsumption and general rules

- We need one extra typing rule stating that if τ is a subtype of σ, then any term of type τ can also regarded as having type σ.

Subtyping: subsumption and general rules

- We need one extra typing rule stating that if τ is a subtype of σ, then any term of type τ can also regarded as having type σ.
  - The rule of subsumption:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

Subtyping: subsumption and general rules

- We need one extra typing rule stating that if τ is a subtype of σ, then any term of type τ can also regarded as having type σ.
  - The rule of subsumption:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

- We will also need set of rules for specifying subtyping relation between types.
  - General rules:

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

$$\frac{\tau <: \rho \qquad \rho <: \sigma}{\tau <: \sigma} \qquad \text{(S-Trans)}$$

Subtyping: rules for records

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

$$\{l_i : \tau_i \,^{i \in 1..n+k}\} \quad <: \quad \{l_i : \tau_i \,^{i \in 1..n}\} \qquad \text{(S-RcdWidth)}$$

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

Subtyping: rules for records

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

$$\{l_i : \tau_i{}^{i \in 1..n+k}\} \quad <: \quad \{l_i : \tau_i{}^{i \in 1..n}\} \qquad \text{(S-RcdWidth)}$$

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

- "Depth subtyping" within fields:

$$\frac{\tau_1 <: \sigma_1 \quad \cdots \quad \tau_n <: \sigma_n}{\{l_i : \tau_i{}^{i \in 1..n}\} \; <: \; \{l_i : \sigma_i{}^{i \in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

Subtyping: rules for records

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

$$\{l_i : \tau_i{}^{i \in 1..n+k}\} \quad <: \quad \{l_i : \tau_i{}^{i \in 1..n}\} \qquad \text{(S-RcdWidth)}$$

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

- "Depth subtyping" within fields:

$$\frac{\tau_1 <: \sigma_1 \quad \cdots \quad \tau_n <: \sigma_n}{\{l_i : \tau_i{}^{i \in 1..n}\} \; <: \; \{l_i : \sigma_i{}^{i \in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

- Permutation of fields:

$$\frac{\{k_j : \tau_j{}^{j \in 1..n}\} \quad \text{is a permutation of} \quad \{l_i : \sigma_i{}^{i \in 1..n}\}}{\{k_j : \tau_j{}^{j \in 1..n}\} \; <: \; \{l_i : \sigma_i{}^{i \in 1..n}\}} \qquad \text{(S-RcdPerm)}$$

Records subtyping: examples

$\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} \ <: \ \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\}}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \quad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \quad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{m{:}\mathrm{Nat}\}}\ \text{(S-Refl)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\}}\ \text{(S-RcdDepth)}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{m{:}\mathrm{Nat}\}}\ \text{(S-Refl)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\}\}}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{}{\{m{:}\mathrm{Nat}\} <: \{m{:}\mathrm{Nat}\}}\ \text{(S-Refl)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\}}\ \text{(S-RcdDepth)}$$

$$\dfrac{\dfrac{}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\}\}}\ \text{(S-RcdWidth)} \qquad \dfrac{\dfrac{}{\{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\} <: \{a{:}\mathrm{Nat}\}}\ \text{(S-RcdWidth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\}\}}\ \text{(S-RcdDepth)}}{\{x : \{a{:}\mathrm{Nat},\ b{:}\mathrm{Nat}\},\ y : \{m{:}\mathrm{Nat}\}\} <: \{x : \{a{:}\mathrm{Nat}\}\}}\ \text{(S-Trans)}$$

Records subtyping: examples

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

Records subtyping: examples

$$\dfrac{\dfrac{}{\vdash (\lambda r : \{x{:}\mathsf{Nat}\}.\ r.x) : \{x{:}\mathsf{Nat}\} \to \mathsf{Nat}} \qquad \dfrac{}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathsf{Nat}\}}}{\vdash (\lambda r : \{x{:}\mathsf{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\} : \mathsf{Nat}}$$

Records subtyping: examples

$$\dfrac{\dfrac{}{r : \{x{:}\mathsf{Nat}\} \vdash r.x : \mathsf{Nat}}}{\vdash (\lambda r : \{x{:}\mathsf{Nat}\}.\ r.x) : \{x{:}\mathsf{Nat}\} \to \mathsf{Nat}} \qquad \dfrac{}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathsf{Nat}\}}$$

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

Records subtyping: examples

$$\dfrac{\dfrac{\dfrac{}{r : \{x{:}\mathsf{Nat}\} \vdash r : \{x{:}\mathsf{Nat}\}}}{r : \{x{:}\mathsf{Nat}\} \vdash r.x : \mathsf{Nat}}}{\vdash (\lambda r : \{x{:}\mathsf{Nat}\}.\ r.x) : \{x{:}\mathsf{Nat}\} \to \mathsf{Nat}} \qquad \dfrac{}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathsf{Nat}\}}$$

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

Records subtyping: examples

$$\dfrac{\dfrac{\dfrac{}{r : \{x{:}\mathrm{Nat}\} \vdash r : \{x{:}\mathrm{Nat}\}}}{r : \{x{:}\mathrm{Nat}\} \vdash r.x : \mathrm{Nat}}}{\vdash (\lambda r : \{x{:}\mathrm{Nat}\}.\ r.x) : \{x{:}\mathrm{Nat}\} \to \mathrm{Nat}} \qquad \dfrac{\dfrac{}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}} \qquad \dfrac{}{\{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\} <: \{x{:}\mathrm{Nat}\}}}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathrm{Nat}\}}$$

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

Records subtyping: examples

$$\dfrac{\dfrac{\dfrac{}{r : \{x{:}\mathrm{Nat}\} \vdash r : \{x{:}\mathrm{Nat}\}}}{r : \{x{:}\mathrm{Nat}\} \vdash r.x : \mathrm{Nat}}}{\vdash (\lambda r : \{x{:}\mathrm{Nat}\}.\ r.x) : \{x{:}\mathrm{Nat}\} \to \mathrm{Nat}} \qquad \dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathrm{Nat}} \qquad \dfrac{}{\vdash 1 : \mathrm{Nat}}}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\}} \qquad \dfrac{}{\{x{:}\mathrm{Nat},\ y{:}\mathrm{Nat}\} <: \{x{:}\mathrm{Nat}\}}}{\vdash \{x = 0,\ y = 1\} : \{x{:}\mathrm{Nat}\}}}{\vdash (\lambda r : \{x{:}\mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\} : \mathrm{Nat}}$$

Variations of record subtyping

- Real languages may sometimes use only a subset 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;
  - ie., no depth subtyping.
- Each class has just one superclass (single inheritance):
  - each class member can be assigned a single index, adding new indexes "on the right" as more members are added in subclasses;
  - ie., no permutation for classes.
- A class may implement multiple interfaces (multiple inheritance of interfaces);
  - ie., permutation is allowed for interfaces.

Subtyping: rules for arrow types and Top

- Arrow types:

$$\frac{\sigma_1 <: \tau_1 \qquad \tau_2 <: \sigma_2}{\tau_1 \to \tau_2 \;<:\; \sigma_1 \to \sigma_2} \qquad \text{(S-Arrow)}$$

  - Note that the subtype relation is contravariant in the left-hand sides of arrows and covariant in the righ-hand sides.

Subtyping: rules for arrow types and Top

- Arrow types:

$$\frac{\sigma_1 <: \tau_1 \qquad \tau_2 <: \sigma_2}{\tau_1 \to \tau_2 \;<:\; \sigma_1 \to \sigma_2} \qquad \text{(S-Arrow)}$$

  - Note that the subtype relation is contravariant in the left-hand sides of arrows and covariant in the righ-hand sides.

- Top type:

$$\tau \;<:\; \mathsf{Top} \qquad \text{(S-Top)}$$

  - Maximal type (cf. Object in Java).
  - In the current setting not necessary, but often included for convenience.

Subtyping: type safety

- Preservation and Progress theorems continue to hold.
  - Preservation: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
  - Progress: If $\vdash e : \tau$, then either e is a value or there exists e' such that $e \rightarrow e'$.
- But proofs are a bit more involved, because the typing relation is no longer syntax directed.

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Proof: By induction on typing derivation.

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-Sub:
  - Assume $\Gamma \vdash e : \sigma$ and $\sigma <: \tau$.

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \tau}{\Gamma \vdash e : \tau} \quad \text{(T-Sub)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-Sub:
  - Assume $\Gamma \vdash e : \sigma$ and $\sigma <: \tau$.
  - By induction hypothesis, $\Gamma \vdash e' : \sigma$.

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \tau}{\Gamma \vdash e : \tau} \text{ (T-Sub)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-Sub:
  - Assume $\Gamma \vdash e : \sigma$ and $\sigma <: \tau$.
  - By induction hypothesis, $\Gamma \vdash e' : \sigma$.
  - By (T-Sub), $\Gamma \vdash e' : \tau$.

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \tau}{\Gamma \vdash e : \tau} \text{(T-Sub)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - By the inversion lemma for evaluation, there are three rules by which $e \twoheadrightarrow e'$ can be derived. Proceed by cases.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - By the inversion lemma for evaluation, there are three rules by which $e \twoheadrightarrow e'$ can be derived. Proceed by cases.
  - Case E-App1:
    - Assume $e' = e_1'\, e_2$ and $e_1 \twoheadrightarrow e_1'$.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$\frac{e_1 \twoheadrightarrow e_1'}{e_1\, e_2 \twoheadrightarrow e_1'\, e_2} \qquad \text{(E-App1)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - By the inversion lemma for evaluation, there are three rules by which $e \twoheadrightarrow e'$ can be derived. Proceed by cases.
  - Case E-App1:
    - Assume $e' = e_1'\, e_2$ and $e_1 \twoheadrightarrow e_1'$.
    - The result follows by IH and (T-App).

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$\frac{e_1 \twoheadrightarrow e_1'}{e_1\, e_2 \twoheadrightarrow e_1'\, e_2} \qquad \text{(E-App1)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - By the inversion lemma for evaluation, there are three rules by which $e \twoheadrightarrow e'$ can be derived. Proceed by cases.
  - Case E-App2:
    - Assume $e_1 = v_1$, $e' = v_1\, e_2'$ and $e_2 \twoheadrightarrow e_2'$.
    - Similar.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$\frac{e_2 \twoheadrightarrow e_2'}{v_1\, e_2 \twoheadrightarrow v_1\, e_2'} \qquad \text{(E-App2)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - Case E-AppAbs:
    - Assume $e_1 = \lambda x{:}\rho.\ e_3$, $e_2 = v_2$ and $e' = e_3[x \mapsto v_2]$.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$(\lambda x{:}\rho.\ e_3)\, v_2 \twoheadrightarrow e_3[x \mapsto v_2] \qquad \text{(E-AppAbs)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - Case E-AppAbs:
    - Assume $e_1 = \lambda x{:}\rho.\ e_3$, $e_2 = v_2$ and $e' = e_3[x \mapsto v_2]$.
    - By inversion lemma for typing relation ...

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$(\lambda x{:}\rho.\ e_3)\, v_2 \twoheadrightarrow e_3[x \mapsto v_2] \qquad \text{(E-AppAbs)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - Case E-AppAbs:
    - Assume $e_1 = \lambda x{:}\rho.\ e_3$, $e_2 = v_2$ and $e' = e_3[x \mapsto v_2]$.
    - By inversion lemma for typing relation ...
      $\sigma <: \rho$ and $\Gamma, x{:}\rho \vdash e_3 : \tau$.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$(\lambda x{:}\rho.\ e_3)\, v_2 \twoheadrightarrow e_3[x \mapsto v_2] \qquad \text{(E-AppAbs)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - Case E-AppAbs:
    - Assume $e_1 = \lambda x{:}\rho.\ e_3$, $e_2 = v_2$ and $e' = e_3[x \mapsto v_2]$.
    - By inversion lemma for typing relation ...
      $\sigma <: \rho$ and $\Gamma, x{:}\rho \vdash e_3 : \tau$.
    - By (T-Sub), $\Gamma \vdash e_2 : \rho$.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$(\lambda x{:}\rho.\ e_3)\, v_2 \twoheadrightarrow e_3[x \mapsto v_2] \qquad \text{(E-AppAbs)}$$

Preservation

- Theorem: If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow e'$, then $\Gamma \vdash e' : \tau$.
- Case T-App:
  - Assume $e = e_1\, e_2$, $\Gamma \vdash e_1 : \sigma \to \tau$ and $\Gamma \vdash e_2 : \sigma$.
  - Case E-AppAbs:
    - Assume $e_1 = \lambda x{:}\rho.\ e_3$, $e_2 = v_2$ and $e' = e_3[x \mapsto v_2]$.
    - By inversion lemma for typing relation ...
      $\sigma <: \rho$ and $\Gamma, x{:}\rho \vdash e_3 : \tau$.
    - By (T-Sub), $\Gamma \vdash e_2 : \rho$.
    - By the substitution lemma $\Gamma \vdash e' : \tau$, and we are done.

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

$$(\lambda x{:}\rho.\ e_3)\, v_2 \twoheadrightarrow e_3[x \mapsto v_2] \qquad \text{(E-AppAbs)}$$

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 \;:\; \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.
- Case T-SUB:
  - Assume $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \sigma$ and $\sigma <: \tau_1 \to \tau_2$.

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.
- Case T-SUB:
  - Assume $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \sigma$ and $\sigma <: \tau_1 \to \tau_2$.
  - We want to say "By the ind. hyp.", but IH doesn't apply (we do not know that σ is an arrow type).

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.
- Case T-SUB:
  - Assume $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \sigma$ and $\sigma <: \tau_1 \to \tau_2$.
  - We want to say "By the ind. hyp.", but IH doesn't apply (we do not know that σ is an arrow type).
  - Need another lemma ...
  - Lemma: If $\sigma <: \tau_1 \to \tau_2$, then $\sigma = \sigma_1 \to \sigma_2$ with $\tau_1 <: \sigma_1$ and $\sigma_2 <: \tau_2$.

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.
- Case T-SUB:
  - Assume $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \sigma$ and $\sigma <: \tau_1 \to \tau_2$.
  - We want to say "By the ind. hyp.", but IH doesn't apply (we do not know that σ is an arrow type).
  - Need another lemma ...
  - Lemma: If $\sigma <: \tau_1 \to \tau_2$, then $\sigma = \sigma_1 \to \sigma_2$ with $\tau_1 <: \sigma_1$ and $\sigma_2 <: \tau_2$.
  - Now IH applies, yielding $\sigma_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \sigma_2$.

Inversion lemma for typing

- Lemma: If $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \tau_1 \to \tau_2$, then $\tau_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \tau_2$.
- Proof by induction on typing derivations.
- Case T-SUB:
  - Assume $\Gamma \vdash \lambda x{:}\rho_1.\, e_2 : \sigma$ and $\sigma <: \tau_1 \to \tau_2$.
  - We want to say "By the ind. hyp.", but IH doesn't apply (we do not know that σ is an arrow type).
  - Need another lemma ...
  - Lemma: If $\sigma <: \tau_1 \to \tau_2$, then $\sigma = \sigma_1 \to \sigma_2$ with $\tau_1 <: \sigma_1$ and $\sigma_2 <: \tau_2$.
  - Now IH applies, yielding $\sigma_1 <: \rho_1$ and $\Gamma, x{:}\rho_1 \vdash e_2 : \sigma_2$.
  - From $\sigma_1 <: \rho_1$ and $\tau_1 <: \sigma_1$, rule (S-TRANS) gives $\tau_1 <: \rho_1$
  - ... and we are done.

Variants

- Types: $\tau ::= \ldots$
  $\mid \langle l_i : \tau_i \,^{i \in 1..n} \rangle$ variant type
- Terms:

  $e ::= \ldots$
  $\mid \langle l = e \rangle$ tagging
  $\mid \mathbf{case}\, e\, \mathbf{of}\, \langle l_i = x_i \rangle \Rightarrow e_i \,^{i \in 1..n}$ case
- Values: $v ::= \ldots$
  $\mid \langle l = v \rangle$ tagged value

Evaluation rules for variants

$$\mathbf{case}\ \langle l_j = v_j \rangle\ \mathbf{of}\ \langle l_i = x_i \rangle \Rightarrow e_i\ ^{i \in 1..n} \quad \rightarrow \quad e_j[x_j \mapsto v_j] \qquad \text{(E-CaseVariant)}$$

$$\frac{e_0 \rightarrow e_0'}{\mathbf{case}\ e_0\ \mathbf{of}\ \langle l_i = x_i \rangle \Rightarrow e_i\ ^{i \in 1..n} \rightarrow \mathbf{case}\ e_0'\ \mathbf{of}\ \langle l_i = x_i \rangle \Rightarrow e_i\ ^{i \in 1..n}} \qquad \text{(E-Case)}$$

$$\frac{e \rightarrow e'}{\langle l = e \rangle \rightarrow \langle l = e' \rangle} \qquad \text{(E-Variant)}$$

Typing rules for variants

$$\frac{\Gamma \vdash e : \langle l_i : \tau_i \,^{i \in 1..n} \rangle \qquad \forall i.\ \Gamma, x_i{:}\tau_i \vdash e_i : \tau}{\Gamma \vdash \mathbf{case}\ e\ \mathbf{of}\ \langle l_i = x_i \rangle \Rightarrow e_i \,^{i \in 1..n} \ :\ \tau} \qquad \text{(T-Case)}$$

$$\frac{\Gamma \vdash e_j : \tau_j}{\Gamma \vdash \langle l_j = e_j \rangle \ :\ \langle l_i : \tau_i \,^{i \in 1..n} \rangle} \qquad \text{(T-Variant)}$$

- Note that here the tagging expression must "know" its type.

Typing rules for variants

$$\frac{\Gamma \vdash e : \langle l_i : \tau_i \,^{i \in 1..n} \rangle \qquad \forall i.\ \Gamma, x_i{:}\tau_i \vdash e_i : \tau}{\Gamma \vdash \mathbf{case}\ e\ \mathbf{of}\ \langle l_i = x_i \rangle \Rightarrow e_i \,^{i \in 1..n} \ :\ \tau} \quad \text{(T-Case)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \langle l_1 = e_1 \rangle \ :\ \langle l_1 : \tau_1 \rangle} \quad \text{(T-Variant)}$$

- With subtyping, we can use singleton variant types for tagging expressions and "add" new variants later.

Subtyping rules for variants

- "Width subtyping":

$$\langle l_i : \tau_i \,^{i \in 1..n} \rangle \quad <: \quad \langle l_i : \tau_i \,^{i \in 1..n+k} \rangle \qquad \text{(S-VariantWidth)}$$

  - Note that the variant type with more tags is a supertype of the variant type with fewer tags.

Subtyping rules for variants

- "Width subtyping":

$$\langle l_i : \tau_i \,^{i \in 1..n} \rangle \quad <: \quad \langle l_i : \tau_i \,^{i \in 1..n+k} \rangle \qquad \text{(S-VariantWidth)}$$

  - Note that the variant type with more tags is a supertype of the variant type with fewer tags.

- "Depth subtyping":

$$\frac{\tau_1 <: \sigma_1 \quad \cdots \quad \tau_n <: \sigma_n}{\langle l_i : \tau_i \,^{i \in 1..n} \rangle \;<:\; \langle l_i : \sigma_i \,^{i \in 1..n} \rangle} \qquad \text{(S-VariantDepth)}$$

- Permutation:

$$\frac{\langle k_j : \tau_j \,^{j \in 1..n} \rangle \quad \text{is a permutation of} \quad \langle l_i : \sigma_i \,^{i \in 1..n} \rangle}{\langle k_j : \tau_j \,^{j \in 1..n} \rangle \;<:\; \langle l_i : \sigma_i \,^{i \in 1..n} \rangle} \qquad \text{(S-VariantPerm)}$$

References

- Types: $\tau ::= \ldots$
  $\mid \mathbf{Ref}\ \tau$ type of references
- Terms: $e ::= \ldots$
  $\mid \mathbf{ref}\ e$ initialized allocation
  $\mid\ !\ e$ dereference
  $\mid e_1 := e_2$ assignment
- For simplicity, we consider here only static semantics (ie. typing) and ignore dynamic semantics (ie. evaluation).

Typing rules for references

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{ref}\, e : \mathbf{Ref}\, \tau} \qquad \text{(T-Ref)}$$

$$\frac{\Gamma \vdash e : \mathbf{Ref}\, \tau}{\Gamma \vdash\!!\, e : \tau} \qquad \text{(T-Deref)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{Ref}\, \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \mathrm{Unit}} \qquad \text{(T-Assign)}$$

Typing rules for references

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{ref}\, e : \mathbf{Ref}\, \tau} \qquad \text{(T-Ref)}$$

$$\frac{\Gamma \vdash e : \mathbf{Ref}\, \tau}{\Gamma \vdash\!! e : \tau} \qquad \text{(T-Deref)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{Ref}\, \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \mathrm{Unit}} \qquad \text{(T-Assign)}$$

$$\frac{\sigma <: \tau \qquad \tau <: \sigma}{\mathbf{Ref}\, \sigma <: \mathbf{Ref}\, \tau} \qquad \text{(S-Ref)}$$

Subtyping of arrays

- Elements of mutable arrays are acting like references.
- Hence, the subtyping rule should be invariant:

$$\frac{\sigma <: \tau \qquad \tau <: \sigma}{\mathbf{Array}\,\sigma <: \mathbf{Array}\,\tau} \qquad \text{(S-ARRAY)}$$

- Interestingly, in Java arrays have covariant subtyping:

$$\frac{\sigma <: \tau}{\mathbf{Array}\,\sigma <: \mathbf{Array}\,\tau} \qquad \text{(S-ARRAYJAVA)}$$

Algorithmic (sub)typing

- In simply typed λ-calculus (without subtyping) typing rules have natural algorithmic reading leading to type checking and type inference procedures.
- For instance, consider the rule for application:

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

- Given a context Γ and some term e in the form $e_1\, e_2$, we can try to find a type for e by
  - finding (recursively) a type for e_1;
  - checking that it has the form $\sigma \to \tau$;
  - finding (recursively) a type for e_2;
  - checking that it is the same as σ.

Algorithmic (sub)typing

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \quad \text{(T-App)}$$

The reason why this works is twofold:

Algorithmic (sub)typing

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

The reason why this works is twofold:

- The set of typing rules is syntax-directed.
  - For any expression of a given shape there is only one rule that can be used to derive typing statements involving expressions that shape.

Algorithmic (sub)typing

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

The reason why this works is twofold:

- The set of typing rules is syntax-directed.
- We can divide the "positions" of the typing relation into input positions (Γ and e) and output positions (τ).

Algorithmic (sub)typing

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-App)}$$

The reason why this works is twofold:

- The set of typing rules is syntax-directed.
- We can divide the "positions" of the typing relation into input positions (Γ and e) and output positions (τ).
  - For the input positions, all metavariables appearing in the premises also appear in the conclusion (so we can calculate inputs to the subgoals from the subexpressions of inputs to the main goal).
  - For the output positions, all metavariables appearing in the conclusion also appear in the premises (so we can calculate outputs of the main goal from the outputs of the subgoals).

Algorithmic (sub)typing

- However, the subtyping relation is clearly not syntax directed.
- For instance, $\tau <: \tau$ can be derived directly using reflexivity, or indirectly using transitivity with subgoals using reflexivity, or ...
- The transitivity rule contains a metavariable in premises which does not appear at all in the conclusion.

$$\frac{\tau <: \rho \qquad \rho <: \sigma}{\tau <: \sigma} \qquad \text{(S-Trans)}$$

  - Hence, we have to guess the value of ρ.
- Record width, depth and permutation rules overlap with each other.

Algorithmic (sub)typing

- Also, the subsumption rule is not syntax-directed.

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

- It is applicable for terms of arbitrary shape.
  - For any term we now have two rules which can be applied.
- The "inputs" of the left-hand subgoal are exactly the same as the inputs to the main goal.
  - May lead to nontermination of the type checking.
- The subtyping relation is not functional.
  - Hence, τ appears both in input and output positions.

Algorithmic (sub)typing

- For algorithmic (sub)typing we need to reformulate typing and subtyping rules to be syntax-directed.
- We also need to prove, that the new "algorithmic" type system is equivalent to the original "declarative" one in an appropriate sense.

Algorithmic subtyping

- All three rules for records (width, depth and permutation) are replaced by the single rule:

$$\frac{\{l_i{}^{i\in 1..n}\} \subseteq \{k_j{}^{j\in 1..m}\} \qquad k_j = l_i \quad \text{implies} \quad \sigma_j <: \tau_i}{\{k_j : \sigma_j{}^{j\in 1..m}\} \;<:\; \{l_i : \tau_i{}^{i\in 1..n}\}} \qquad \text{(S-RCD)}$$

- Lemma: If $\sigma <: \tau$ is provable with the 3 separate rules for width, depth and permutation, it is provable with just (S-RCD) (and vice versa).

Algorithmic subtyping

- Lemma: $\tau <: \tau$ can be derived for every type τ without using (S-Refl).

Algorithmic subtyping

- Lemma: $\tau <: \tau$ can be derived for every type τ without using (S-Refl).
- Note, however, that if we add base types (like Bool or Int), we also need to add the corresponding reflection rules:

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

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

Algorithmic subtyping

- Lemma: $\tau <: \tau$ can be derived for every type τ without using (S-Refl).
- Note, however, that if we add base types (like Bool or Int), we also need to add the corresponding reflection rules:

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

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

- Lemma: If $\sigma <: \tau$ is derivable, then it can be derived without using (S-Trans).

Algorithmic subtyping

- Lemma: $\tau <: \tau$ can be derived for every type τ without using (S-Refl).
- Note, however, that if we add base types (like Bool or Int), we also need to add the corresponding reflection rules:

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

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

- Lemma: If $\sigma <: \tau$ is derivable, then it can be derived without using (S-Trans).
- Hence, we can drop (S-Refl) and (S-Trans) rules from our system.

Algorithmic subtyping rules

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

$$\frac{\tau_1 <: \sigma_1 \qquad \sigma_2 <: \tau_2}{\sigma_1 \to \sigma_2 \;<:\; \tau_1 \to \tau_2} \qquad \text{(S-Arrow)}$$

$$\frac{\{l_i{}^{i\in 1..n}\} \subseteq \{k_j{}^{j\in 1..m}\} \qquad k_j = l_i \quad \text{implies} \quad \sigma_j <: \tau_i}{\{k_j : \sigma_j{}^{j\in 1..m}\} \;<:\; \{l_i : \tau_i{}^{i\in 1..n}\}} \qquad \text{(S-Rcd)}$$

Algorithmic typing

- For typing relation, we have just one problematic rule:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

Algorithmic typing

- For typing relation, we have just one problematic rule:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

- Where is this rule really needed?
- Clearly, we need subsumption for applications. Eg. the term $(\lambda r : \{x : \mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\}$ is not typable without using subsumption.

Algorithmic typing

- For typing relation, we have just one problematic rule:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

- Where is this rule really needed?
- Clearly, we need subsumption for applications. Eg. the term $(\lambda r : \{x : \mathrm{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\}$ is not typable without using subsumption.
- It turns out, that this is the only palce where subsumption is essential.
  - Why? Because (T-App) is the only rule where types must match.

Algorithmic typing

- For typing relation, we have just one problematic rule:

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \sigma}{\Gamma \vdash e : \sigma} \qquad \text{(T-Sub)}$$

- Where is this rule really needed?
- Clearly, we need subsumption for applications. Eg. the term $(\lambda r : \{x : \text{Nat}\}.\ r.x)\ \{x = 0,\ y = 1\}$ is not typable without using subsumption.
- It turns out, that this is the only palce where subsumption is essential.
  - Why? Because (T-App) is the only rule where types must match.
- Every other use of subsumption can be "pushed out".
  - We can rewrite any derivation $\Gamma \vdash e : \tau$ into a special form where (T-Sub) appears only just before an application or at the very end of derivation.

Pushing subsumption out

Case (T-Abs):

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma,\, x{:}\tau_1 \vdash e : \sigma_2} \qquad \dfrac{\vdots}{\sigma_2 <: \tau_2}}{\Gamma,\, x{:}\tau_1 \vdash e : \tau_2}\ \text{(T-Sub)}}{\Gamma \vdash \lambda x{:}\tau_1.\, e : \tau_1 \to \tau_2}\ \text{(T-Abs)}$$

Pushing subsumption out

Case (T-Abs):

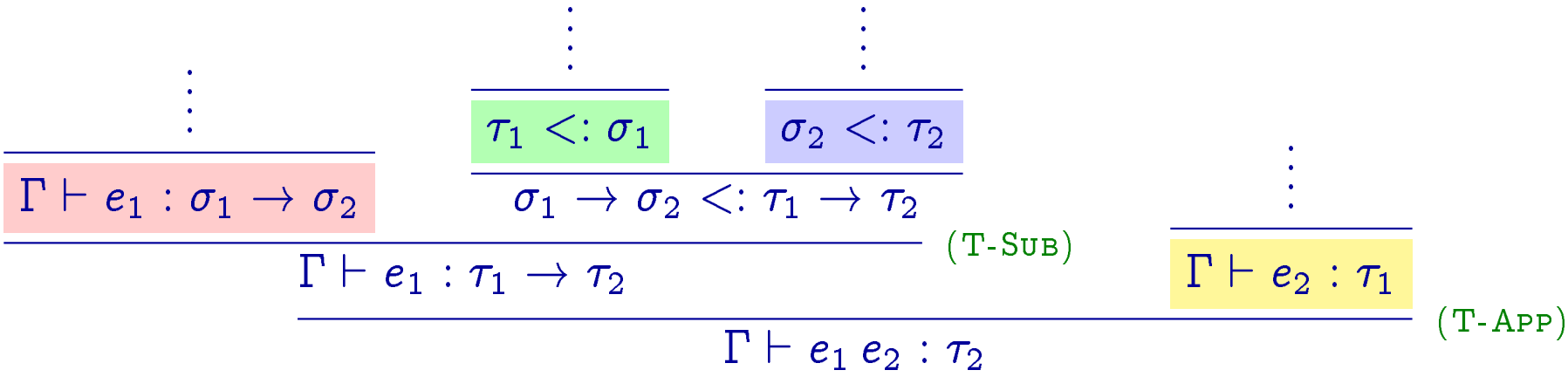
$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma,\, x{:}\tau_1 \vdash e : \sigma_2} \qquad \dfrac{\vdots}{\sigma_2 <: \tau_2}}{\Gamma,\, x{:}\tau_1 \vdash e : \tau_2}\ (\text{T-Sub})}{\Gamma \vdash \lambda x{:}\tau_1 . e : \tau_1 \to \tau_2}\ (\text{T-Abs})$$

becomes

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma,\, x{:}\tau_1 \vdash e : \sigma_2}}{\Gamma \vdash \lambda x{:}\tau_1 . e : \tau_1 \to \sigma_2}\ (\text{T-Abs}) \qquad \dfrac{\dfrac{}{\tau_1 <: \tau_1} \qquad \dfrac{\vdots}{\sigma_2 <: \tau_2}}{\tau_1 \to \sigma_2 <: \tau_1 \to \tau_2}}{\Gamma \vdash \lambda x{:}\tau_1 . e : \tau_1 \to \tau_2}\ (\text{T-Sub})$$

Pushing subsumption out

Case (T-App):



Pushing subsumption out

Case (T-App):

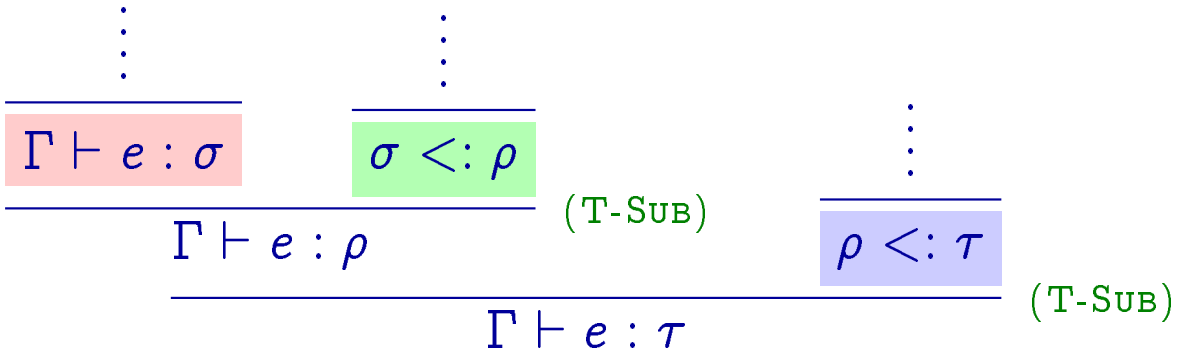
$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e_1 : \sigma_1 \to \sigma_2} \quad \dfrac{\dfrac{\vdots}{\tau_1 <: \sigma_1} \quad \dfrac{\vdots}{\sigma_2 <: \tau_2}}{\sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2}}{\Gamma \vdash e_1 : \tau_1 \to \tau_2} \text{(T-Sub)} \qquad \dfrac{\vdots}{\Gamma \vdash e_2 : \tau_1}}{\Gamma \vdash e_1\ e_2 : \tau_2} \text{(T-App)}$$

becomes

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e_1 : \sigma_1 \to \sigma_2} \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash e_2 : \tau_1} \quad \dfrac{\vdots}{\tau_1 <: \sigma_1}}{\Gamma \vdash e_2 : \sigma_1} \text{(T-Sub)}}{\Gamma \vdash e_1\ e_2 : \sigma_2} \text{(T-App)} \qquad \dfrac{\vdots}{\sigma_2 <: \tau_2}}{\Gamma \vdash e_1\ e_2 : \tau_2} \text{(T-Sub)}$$

Pushing subsumption out

Case (T-Sub):



Pushing subsumption out

Case (T-Sub):

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e : \sigma} \qquad \dfrac{\dfrac{\vdots}{\sigma <: \rho} \qquad \dfrac{\vdots}{\rho <: \tau}}{\sigma <: \tau}\ \text{(T-Sub)}}{\Gamma \vdash e : \tau}\ \text{(T-Sub)}$$

becomes

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e : \sigma} \qquad \dfrac{\dfrac{\vdots}{\sigma <: \rho} \qquad \dfrac{\vdots}{\rho <: \tau}}{\sigma <: \tau}\ \text{(T-Sub)}}{\Gamma \vdash e : \tau}$$

Algorithmic typing

- Hence, we can form an "algorithmic version" of the type system by replacing (T-App) and (T-Sub) rules with the combined one:

$$\frac{\Gamma \vdash e_1 : \rho \to \tau \qquad \Gamma \vdash e_2 : \sigma \qquad \sigma <: \rho}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-AppSub)}$$

Algorithmic typing

- Hence, we can form an "algorithmic version" of the type system by replacing (T-App) and (T-Sub) rules with the combined one:

$$\frac{\Gamma \vdash e_1 : \rho \to \tau \qquad \Gamma \vdash e_2 : \sigma \qquad \sigma <: \rho}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-AppSub)}$$

- Lemma (Soundness): If $\Gamma \vdash e : \tau$ is derivable in the "algorithmic version", then it is also derivable in the "declarative version" of the type system.

Algorithmic typing

- Hence, we can form an "algorithmic version" of the type system by replacing (T-App) and (T-Sub) rules with the combined one:

$$\frac{\Gamma \vdash e_1 : \rho \to \tau \qquad \Gamma \vdash e_2 : \sigma \qquad \sigma <: \rho}{\Gamma \vdash e_1\, e_2 : \tau} \qquad \text{(T-AppSub)}$$

- Lemma (Soundness): If $\Gamma \vdash e : \tau$ is derivable in the "algorithmic version", then it is also derivable in the "declarative version" of the type system.
- Lemma (Completeness): If $\Gamma \vdash e : \tau$ is derivable in the "declarative version", then there exists σ, s.t., $\sigma <: \tau$ and $\Gamma \vdash e : \sigma$ is derivable in the "algorithmic version" of the type system.

Algorithmic typing

- If we have conditionals or case expressions, we need additional machinery to support algorithmic subtyping.

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 : \tau}$$

- Using subsumption, one may give different subtypes of τ for two branches.

Algorithmic typing

- If we have conditionals or case expressions, we need additional machinery to support algorithmic subtyping.

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 : \tau}$$

- Using subsumption, one may give different subtypes of τ for two branches.
- One may try to incorporate the subsumption into the rule above as follows:

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \rho \quad \sigma <: \tau \quad \rho <: \tau}{\Gamma \vdash \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 : \tau}$$

- But this is not satisfactory, as it doesn't guarantee that τ is minimal.

Algorithmic typing

- A type τ is the join of two types σ_1 and σ_2 (written as $\tau = \sigma_1 \vee \sigma_2$), if
  - $\sigma_1 <: \tau$;
  - $\sigma_2 <: \tau$;
  - for all ρ, if $\sigma_1 <: \rho$ and $\sigma_2 <: \rho$ then $\tau <: \rho$.

Algorithmic typing

- A type τ is the join of two types σ_1 and σ_2 (written as $\tau = \sigma_1 \vee \sigma_2$), if
  - $\sigma_1 <: \tau$;
  - $\sigma_2 <: \tau$;
  - for all ρ, if $\sigma_1 <: \rho$ and $\sigma_2 <: \rho$ then $\tau <: \rho$.
- Lemma: For any pair of types σ_1 and σ_2, there is some type τ such that $\tau = \sigma_1 \vee \sigma_2$.

Algorithmic typing

- A type τ is the join of two types σ_1 and σ_2 (written as $\tau = \sigma_1 \vee \sigma_2$), if
  - $\sigma_1 <: \tau$;
  - $\sigma_2 <: \tau$;
  - for all ρ, if $\sigma_1 <: \rho$ and $\sigma_2 <: \rho$ then $\tau <: \rho$.
- Lemma: For any pair of types σ_1 and σ_2, there is some type τ such that $\tau = \sigma_1 \vee \sigma_2$.
- Algorithmic rule for conditionals:

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \rho \quad \sigma \vee \rho = \tau}{\Gamma \vdash \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 : \tau} \qquad \text{(T-If)}$$