510: Programming Languages

Subtyping

David Walker

Type Checking with Subtyping

It is no longer obvious how to type check!

- Rules are no longer syntax-directed. The subsumption rule always applies.

$$\frac{\Gamma \vdash e : \sigma \quad \sigma <: \tau}{\Gamma \vdash e : \tau}$$

- Must decide subtype relation.

Type Checking

Here's why things are complicated.

Suppose that f : $\texttt{float}{\rightarrow}\texttt{int}$ and e : $\texttt{int}$. Then we may show that $\texttt{apply(}f,e\texttt{)}$: $\texttt{float}$ in two different ways:

1. Demote the function by promoting its domain: By subsumption f : $\texttt{int}{\rightarrow}\texttt{float}$, so $\texttt{apply(}f,e\texttt{)}$: $\texttt{float}$.

2. Demote the argument: By subsumpion e : $\texttt{float}$, so $\texttt{apply(}f,e\texttt{)}$: $\texttt{int}$, and so $\texttt{apply(}f,e\texttt{)}$: $\texttt{float}$.

Type Checking

Two key results:

- **Principal typing**: every expression as a **principal type** that can be computed in a syntax-directed manner.

- **Decidability of subtyping**: given σ and τ we can decide whether or not σ <: τ.

Type Checking for MinML

Idea:

- Extract the unique type of an expression in a context.

- Check that it is the desired type.

Type synthesis relation: $\Gamma \vdash e \Rightarrow \tau$.

- A partial function of Γ and e.

- Defined by induction on the structure of e.

Type Checking for MinML$_{<:}$

Types are no longer unique, because of subsumption! But each expression has a principal type.

The type τ_0 is principal for Γ and e if

- $\Gamma \vdash e : \tau_0$.
- If $\Gamma \vdash e : \tau$, then τ_0 <: τ.

Type Checking for MinML$_{<:}$

To determine whether or not $\Gamma \vdash e : \tau$,

- Find the principal type τ_0 of e in Γ.
- Check whether τ_0 <: τ.

If principal types are lacking, this approach does not work!

Computing Principal Types

Define the **type synthesis** judgement $\Gamma \vdash e \Rightarrow \tau$ by the following **syntax-directed** rules.

$$\frac{}{\Gamma \vdash x \Rightarrow \Gamma(x)}$$

$$\frac{\Gamma[f{:}\tau_1{\rightarrow}\tau_2][x{:}\tau_1] \vdash e \Rightarrow \sigma_2 \quad \sigma_2 \texttt{ <: } \tau_2}{\Gamma \vdash \texttt{fun}\, f\ \texttt{(}x\texttt{:}\tau_1\texttt{)}\texttt{:}\tau_2 \texttt{ = } e \Rightarrow \tau_1{\rightarrow}\tau_2}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_2{\rightarrow}\tau \quad \Gamma \vdash e_2 \Rightarrow \sigma_2 \quad \sigma_2 \texttt{ <: } \tau_2}{\Gamma \vdash \texttt{apply(}e_1, e_2\texttt{)} \Rightarrow \tau}$$

Computing Principal Types

Conditionals are problematic.

$$\frac{\begin{array}{cc} \Gamma \vdash e \Rightarrow \sigma & \sigma \ \texttt{<:}\ \texttt{bool} \\ \Gamma \vdash e_1 \Rightarrow \tau_1 & \Gamma \vdash e_2 \Rightarrow \tau_2 \\ \tau_1 \ \texttt{<:}\ \tau & \tau_2 \ \texttt{<:}\ \tau \end{array}}{\Gamma \vdash \texttt{if}_\tau\, e\, \texttt{then}\, e_1\, \texttt{else}\, e_2\, \texttt{fi} \Rightarrow \tau}$$

We assume given the type of the **`then`** and **`else`** branches!

Computing Principal Types

Alternatively, yield the **join** (least upper bound), whenever it exists.

$$\frac{\Gamma \vdash e \Rightarrow \sigma \quad \sigma \mathtt{<:}\ \mathtt{bool} \quad \Gamma \vdash e_1 \Rightarrow \tau_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2}{\Gamma \vdash \mathtt{if}_\tau\, e\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \Rightarrow \tau_1 \vee \tau_2}$$

For MinML with arithmetic subtyping, the required joins exist.

For Java they do not! It requires either $\tau_1 \mathtt{<:}\ \tau_2$ or **vice versa**.

Computing Principal Types

Theorem 1

1. *(Soundness) If* $\Gamma \vdash e \Rightarrow \tau$*, then* $\Gamma \vdash e : \tau$.

2. *(Completeness) If* $\Gamma \vdash e : \tau$*, then there exists* τ_0 *such that* $\Gamma \vdash e \Rightarrow \tau_0$ *and* τ_0 <: τ.

Completeness Proof (Sketch)

Suppose that $\Gamma \vdash e : \tau$ because $\Gamma \vdash e : \sigma$ and σ <: τ.

By induction we have that $\Gamma \vdash e \Rightarrow \sigma'$ with σ' <: σ.

By transitivity of subtyping, σ' <: τ.

Hence $\Gamma \vdash e \Rightarrow \sigma'$ with σ' <: τ.

Completeness Proof (Sketch)

Suppose that $\Gamma \vdash \texttt{apply}(e_1, e_2) : \tau$ by a use of the application rule:

1. $\Gamma \vdash e_1 : \tau_2 \rightarrow \tau$

2. $\Gamma \vdash e_2 : \tau_2$

Completeness Proof (Sketch)

By induction we have

1. $\Gamma \vdash e_1 \Rightarrow \sigma_1$ with σ_1 <: $\tau_2 \rightarrow \tau$.

2. $\Gamma \vdash e_2 \Rightarrow \sigma_2$ with σ_2 <: τ_2.

Completeness Proof (Sketch)

Now we need a **subtyping lemma** that tells us $\sigma_1 = \sigma_2' \to \sigma'$ with τ_2 `<:` σ_2' and σ' `<:` τ.

By transitivity of subtyping, σ_2 `<:` σ_2'.

Hence $\Gamma \vdash \texttt{apply(}e_1, e_2\texttt{)} \Rightarrow \sigma'$.

Since σ' `<:` τ, this completes the proof.

Subtyping Lemma

Lemma 2 (Subtyping, function types)

1. *If* $\sigma_1 \rightarrow \sigma_2$ `<:` τ*, then* $\tau = \tau_1 \rightarrow \tau_2$ *with* τ_1 `<:` σ_1 *and* σ_2 `<:` τ_2*.*

2. *If* σ `<:` $\tau_1 \rightarrow \tau_2$*, then* $\sigma = \sigma_1 \rightarrow \sigma_2$ *with* τ_1 `<:` σ_1 *and* σ_2 `<:` τ_2*.*

Proof: ?

Lemma 3 (Subtyping, numbers)

1. *If* `float <:` τ *then* $\tau =$ `float`*.*

2. *If* `int <:` τ *then* $\tau =$ `float` *or* $\tau =$ `int`*.*

3. *If* τ `<: float` *then* $\tau =$ `float` *or* $\tau =$ `int`*.*

4. *If* τ `<: int` *then* $\tau =$ `float`*.*

Proof: ?

Type Checking for MinML$_{<:}$

Since the type synthesis rules are syntax-directed, we may find the principal type of e in Γ.

- Proceed by induction on the structure of e.

- Apply the only synthesis rule that could apply.

However, we still need a "subroutine" for subtyping.

Subtype Checking

How to decide whether or not σ <: τ?

- Attempt to "run the rules backward".

- All rules work fine, **except** transitivity.

Problem: how to find the mediating type?

- σ <: τ if **there exists** ρ such that σ <: ρ <: τ.

- No information to guide the choice.

Subtype Checking

Strategy:

- Show that reflexivity can be restricted to base types. That way we need only check it at the base case.
- Show that transitivity is **admissible**: we can ignore it entirely!

Restricted Reflexivity

If we replace the general reflexivity rule by these two rules

$$\frac{}{\texttt{int} <: \texttt{int}} \qquad \frac{}{\texttt{float} <: \texttt{float}}$$

then the subtype relation is **still** reflexive.

The proof is an easy induction on the structure of types.

Admissibility of Transitivity

The rule of transitivity of subtyping is **admissible**, meaning that it may be omitted without changing the subtyping relation.

More precisely, let $<:^-$ be the subtype relation defined by omitting the rule of transitivity.

Clearly, if $\sigma <:^- \tau$, then $\sigma <: \tau$.

Admissibility of Transitivity

We prove that if σ <: τ, then σ <:$^-$ τ. The proof is by induction on the derivation of σ <: τ.

The only interesting case is the rule of transitivity; the remaining cases follow directly by induction.

Inductively, we may assume that σ <:$^-$ ρ and ρ <:$^-$ τ. We are to show that σ <:$^-$ τ.

We proceed by case analysis on the proofs of σ <:$^-$ ρ and ρ <:$^-$ τ, transforming these into a proof of σ <:$^-$ τ.

Admissibility of Transitivity

Case 1:

$$\dfrac{\dfrac{}{\texttt{int} <:^{-} \texttt{float}} \quad \dfrac{\vdots}{\texttt{float} <:^{-} \tau}}{\texttt{int} <: \tau}$$

Subtyping lemma: if $\texttt{float} <:^{-} \tau$, then $\tau = \texttt{float}$.

Therefore, the left-hand sub-derivation is the desired derivation.

Admissibility of Transitivity

Case 2:

$$\dfrac{\dfrac{\vdots}{\tau \mathrel{\texttt{<:}^-} \texttt{int}} \quad \dfrac{}{\texttt{int} \mathrel{\texttt{<:}^-} \texttt{float}}}{\tau \mathrel{\texttt{<:}} \texttt{float}}$$

Subtyping lemma: if $\tau \mathrel{\texttt{<:}^-} \texttt{int}$, then $\tau = \texttt{int}$.

Therefore the right-hand sub-derivation is the desired derivation.

Admissibility of Transitivity

Case 3:

$$\frac{\begin{matrix}\vdots \\ \sigma_1 \to \sigma_2 \mathrel{<:^-} \rho\end{matrix} \qquad \begin{matrix}\vdots \\ \rho \mathrel{<:^-} \tau_1 \to \tau_2\end{matrix}}{\sigma_1 \to \sigma_2 \mathrel{<:} \tau_1 \to \tau_2}$$

Lemma If $\sigma_1 \to \sigma_2 \mathrel{<:^-} \rho \mathrel{<:^-} \tau_1 \to \tau_2$, then $\rho = \rho_1 \to \rho_2$, with $\tau_1 \mathrel{<:^-} \rho_1 \mathrel{<:^-} \sigma_1$ and $\sigma_2 \mathrel{<:^-} \rho_2 \mathrel{<:^-} \tau_2$.

Admissibility of Transitivity

Therefore we have

$$\dfrac{\dfrac{\dfrac{\vdots}{\rho_1 \mathrel{<:^-} \sigma_1} \quad \dfrac{\vdots}{\sigma_2 \mathrel{<:^-} \rho_2}}{\sigma_1{\to}\sigma_2 \mathrel{<:^-} \rho_1{\to}\rho_2} \quad \dfrac{\dfrac{\vdots}{\tau_1 \mathrel{<:^-} \rho_1} \quad \dfrac{\vdots}{\rho_2 \mathrel{<:^-} \tau_2}}{\rho_1{\to}\rho_2 \mathrel{<:^-} \tau_1{\to}\tau_2}}{\sigma_1{\to}\sigma_2 \mathrel{<:} \tau_1{\to}\tau_2}$$

Hence, by induction we have

$$\dfrac{\dfrac{\vdots}{\tau_1 \mathrel{<:^-} \sigma_1} \quad \dfrac{\vdots}{\sigma_2 \mathrel{<:^-} \tau_2}}{\sigma_1{\to}\sigma_2 \mathrel{<:^-} \tau_1{\to}\tau_2}$$

Admissibility of Transitivity

Finally:

$$\frac{\overline{\texttt{int} <:^{-} \texttt{int}} \quad \overline{\texttt{int} <:^{-} \texttt{int}}}{\texttt{int} <: \texttt{int}}$$

$$\frac{\overline{\texttt{float} <:^{-} \texttt{float}} \quad \overline{\texttt{float} <:^{-} \texttt{float}}}{\texttt{float} <: \texttt{float}}$$

$$\frac{\overline{\texttt{int} <:^{-} \texttt{float}} \quad \overline{\texttt{float} <:^{-} \texttt{float}}}{\texttt{int} <: \texttt{float}}$$

$$\frac{\overline{\texttt{int} <:^{-} \texttt{int}} \quad \overline{\texttt{int} <:^{-} \texttt{float}}}{\texttt{int} <: \texttt{float}}$$

Subtype Checking

Thus the subtyping relation may be defined by the following rules alone:

$$\frac{}{\texttt{int <: float}} \qquad \frac{}{\texttt{int <: int}} \qquad \frac{}{\texttt{float <: float}}$$

$$\frac{\tau_1 \texttt{ <: } \sigma_1 \quad \sigma_2 \texttt{ <: } \tau_2}{\sigma_1 \to \sigma_2 \texttt{ <: } \tau_1 \to \tau_2}$$

Subtype Checking

To decide whether or not σ `<:` τ:

1. If $\sigma = \tau$, say "yes".

2. If $\sigma =$ `int` and either $\tau =$ `int` or $\tau =$ `float`, say "yes".

3. If $\sigma = \sigma_1 \rightarrow \sigma_2$, $\tau = \tau_1 \rightarrow \tau_2$, then recursively check whether τ_1 `<:` σ_1 and σ_2 `<:` τ_2, saying "yes" iff both say "yes".

4. Otherwise say "no".

Subtype Checking

Theorem 4

This algorithm yields “yes” or “no” according to whether or not σ `<:` τ.

Summary

The extension of MinML with arithmetic subtyping is

- Safe, because we can prove progress and preservation.
- Decidable, because we can use type synthesis to compute the principal type and use the decidability of subtyping.