

Algorithmic Subtyping

Lecture 16

CS 565

4/3/09

Issues

- Subtyping rules are not syntax-directed
 - Main reason is subsumption rule:

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

- Unlike other rules, applies to “bare” term e
- Transitivity has similar issues:

$$\frac{\sigma <: \tau, \tau <: \theta}{\sigma <: \theta}$$

Transitivity also mentions τ that does not appear in the conclusion

Goals

- Need an algorithm that checks subtype satisfiability efficiently
 - Use the structure of types
 - To implement such a procedure, we cannot rely on transitivity rules, and must restrict when we use subsumption.

Transitivity

- Eliminate transitivity by “bundling” depth, width, and permutation rules on records:

$$\{l_i \mid i \in 1 \dots n\} \subseteq \{k_j \mid j \in 1 \dots m\}$$

$$\frac{k_j = l_i \text{ implies } \sigma_j \prec \tau_i}{\{k_j : \sigma_j \mid j \in 1 \dots m\} \prec \{l_i : \tau_i \mid i \in 1 \dots n\}}$$

$$\{k_j : \sigma_j \mid j \in 1 \dots m\} \prec \{l_i : \tau_i \mid i \in 1 \dots n\}$$

Algorithmic Subtyping

subtype(σ, τ) =

if $\tau = \text{Top}$ then true

else if $\sigma = \sigma_1 \rightarrow \sigma_2$ and $\tau = \tau_1 \rightarrow \tau_2$

then subtype(τ_1, σ_1) and

subtype(σ_2, τ_2)

else if $\sigma = \{k_j : \sigma_j \mid j \in 1 \dots m\}$ and

$\tau = \{l_i : \tau_i \mid i \in 1 \dots n\}$

then $\{l_i \mid i \in 1 \dots n\} \subseteq \{k_j \mid j \in 1 \dots m\}$ and for all i , there is

a j such that $k_j = l_i$ and subtype(σ_j, τ_i)

Properties

Algorithm is

- sound (every statement declared true by the algorithm can be derivable from the declarative rules)
- complete (every statement derivable from the declarative rules will be declared true by the algorithm)
- finite (algorithm returns true or false after a finite amount of time)

Haven't addressed the subsumption rule yet, but this rule occurs in the typing relation:

subsumption bridges the gap between the types expected by functions and the actual types of the arguments:

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

Need subsumption to "lift" the record $\{x:\text{Nat}, y:\text{Bool}\}$ to just $\{x:\text{Nat}\}$

Typing Applications

$$\Gamma \vdash e1 : \tau1 \rightarrow \tau2 \quad \Gamma \vdash e2 : \tau3$$
$$\frac{\tau3 \prec: \tau1}{\Gamma \vdash (e1 \ e2) : \tau2}$$
$$\Gamma \vdash (e1 \ e2) : \tau2$$

Typing rule syntax-directed, and subsumption rule directly incorporated within it.

Straightforward to define algorithmic typing relation (see page 217). It is the least relation closed under the rules given. Can now show soundness by induction on algorithmic typing derivations:

$$\text{If } \Gamma \mapsto e : \tau \text{ then } \Gamma \vdash e : \tau$$

Completeness

- Ordinary typing relations can assign potentially many types to a term.
 - Algorithmic typing assigns at most one.
 - Assigns a minimal type
 - If $\Gamma \vdash e : \tau$ then $\Gamma \mapsto e : \sigma$ for some $\sigma \prec \tau$
 - Every typable term in the declarative system has a minimal type.
-

Joins

- A type β is called join of a pair of types σ and τ written $\sigma \vee \tau$ if
 - $\sigma <: \beta$ and $\tau <: \beta$
 - for all types α if $\sigma <: \alpha$, and $\tau <: \alpha$, then $\beta <: \alpha$
 - For every pair of types σ and τ there is some type $\beta = \sigma \vee \tau$
 - Straightforward to give an algorithmic rule for conditionals:

$$\Gamma \vdash e1 : \text{Bool} \quad \Gamma \vdash e2 : \sigma \quad \Gamma \vdash e3 : \tau$$
$$\sigma \vee \tau = \beta$$

$$\Gamma \vdash \text{if } e1 \text{ then } e2 \text{ else } e3 : \beta$$
