CIS 500
Software Foundations
Fall 2006

November 13

Metatheory of Subtyping

Syntax-directed rules

In the simply typed lambda-calculus (without subtyping), each rule can be "read from bottom to top" in a straightforward way.

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

If we are given some Γ and some t of the form $t_1\ t_2$, we can try to find a type for t by

1. finding (recursively) a type for t_1
2. checking that it has the form $T_{11}{\to}T_{12}$
3. finding (recursively) a type for t_2
4. checking that it is the same as T_{11}

Technically, the reason this works is that we can divide the "positions" of the typing relation into *input positions* (Γ and t) and *output positions* (T).

- 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 conclusions also appear in the premises (so we can calculate outputs from the main goal from the outputs of the subgoals)

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

Syntax-directed sets of rules

The second important point about the simply typed lambda-calculus is that the *set* of typing rules is syntax-directed, in the sense that, for every "input" Γ and t, there one rule that can be used to derive typing statements involving t.

E.g., if t is an application, then we must proceed by trying to use T-App. If we succeed, then we have found a type (indeed, the unique type) for t. If it fails, then we know that t is not typable.

$\longrightarrow$ no backtracking!

Non-syntax-directedness of typing

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 give a type to terms of a given shape (the old one plus T-Sub)

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

2. Worse yet, the new rule T-Sub itself is not syntax directed: the inputs to the left-hand subgoal are exactly the same as the inputs to the main goal!
   (If we translated the typing rules naively into a typechecking function, the case corresponding to T-Sub would cause divergence.)

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.
2. The transitivity rule

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

   is badly non-syntax-directed: the premises contain a metavariable (in an "input position") that does not appear at all in the conclusion.

   To implement this rule naively, we'd have to *guess* a value for $\mathtt{U}$!

What to do?

What to do?

1. Observation: We don't *need* 1000 ways to prove a given typing or subtyping statement — one is enough.

   $\longrightarrow$ Think more carefully about the typing and subtyping systems to see where we can get rid of excess flexibility
2. Use the resulting intuitions to formulate new "algorithmic" (i.e., syntax-directed) typing and subtyping relations
3. Prove that the algorithmic relations are "the same as" the original ones in an appropriate sense.

Developing an algorithmic subtyping relation

Subtype relation

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

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

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

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

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

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

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

Issues

For a given subtyping statement, there are multiple rules that could be used last in a derivation.

1. The conclusions of S-RcdWidth, S-RcdDepth, and S-RcdPerm overlap with each other.
2. S-Refl and S-Trans overlap with every other rule.

Step 1: simplify record subtyping

Idea: combine all three record subtyping rules into one "macro rule" that captures all of their effects

$$\frac{\{\mathtt{l}_i{}^{\,i\in 1..n}\} \subseteq \{\mathtt{k}_j{}^{\,j\in 1..m}\} \qquad \mathtt{k}_j = \mathtt{l}_i \text{ implies } \mathtt{S}_j <: \mathtt{T}_i}{\{\mathtt{k}_j{:}\mathtt{S}_j{}^{\,j\in 1..m}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i{}^{\,i\in 1..n}\}} \qquad \text{(S-RCD)}$$

Simpler subtype relation

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

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

$$\frac{\{\mathtt{l}_i{}^{\,i\in 1..n}\} \subseteq \{\mathtt{k}_j{}^{\,j\in 1..m}\} \qquad \mathtt{k}_j = \mathtt{l}_i \text{ implies } \mathtt{S}_j <: \mathtt{T}_i}{\{\mathtt{k}_j{:}\mathtt{S}_j{}^{\,j\in 1..m}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i{}^{\,i\in 1..n}\}} \qquad \text{(S-RCD)}$$

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

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

Step 2: Get rid of reflexivity

Observation: S-REFL is unnecessary.

Lemma: $\mathtt{S} <: \mathtt{S}$ can be derived for every type $\mathtt{S}$ without using S-REFL.

Even simpler subtype relation

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

$$\frac{\{\mathtt{l}_i{}^{\,i\in 1..n}\} \subseteq \{\mathtt{k}_j{}^{\,j\in 1..m}\} \qquad \mathtt{k}_j = \mathtt{l}_i \text{ implies } \mathtt{S}_j <: \mathtt{T}_i}{\{\mathtt{k}_j{:}\mathtt{S}_j{}^{\,j\in 1..m}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i{}^{\,i\in 1..n}\}} \qquad \text{(S-RCD)}$$

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

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

Step 3: Get rid of transitivity

Observation: S-TRANS is unnecessary.

Lemma: If $\mathtt{S} <: \mathtt{T}$ can be derived, then it can be derived without using S-TRANS.

"Algorithmic" subtype relation

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

$$\frac{\mapsto \mathtt{T}_1 <: \mathtt{S}_1 \qquad \mapsto \mathtt{S}_2 <: \mathtt{T}_2}{\mapsto \mathtt{S}_1 \to \mathtt{S}_2 <: \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(SA-ARROW)}$$

$$\frac{\{\mathtt{l}_i{}^{\,i\in 1..n}\} \subseteq \{\mathtt{k}_j{}^{\,j\in 1..m}\} \qquad \text{for each } \mathtt{k}_j = \mathtt{l}_i,\ \mapsto \mathtt{S}_j <: \mathtt{T}_i}{\mapsto \{\mathtt{k}_j{:}\mathtt{S}_j{}^{\,j\in 1..m}\} <: \{\mathtt{l}_i{:}\mathtt{T}_i{}^{\,i\in 1..n}\}} \qquad \text{(SA-RCD)}$$

Soundness and completeness

Theorem: $\mathtt{S} <: \mathtt{T}$ iff $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$.

Proof: *(Homework)*

Terminology:

- The algorithmic presentation of subtyping is *sound* with respect to the original if $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$ implies $\mathtt{S} <: \mathtt{T}$. (Everything validated by the algorithm is actually true.)
- The algorithmic presentation of subtyping is *complete* with respect to the original if $\mathtt{S} <: \mathtt{T}$ implies $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$. (Everything true is validated by the algorithm.)

Subtyping Algorithm (pseudo-code)

The algorithmic rules can be translated directly into code:

subtype$(\mathtt{S}, \mathtt{T})$ =

- if $\mathtt{T} = \mathtt{Top}$, then *true*
- else if $\mathtt{S} = \mathtt{S}_1 \rightarrow \mathtt{S}_2$ and $\mathtt{T} = \mathtt{T}_1 \rightarrow \mathtt{T}_2$
  - then $subtype(\mathtt{T}_1, \mathtt{S}_1) \wedge subtype(\mathtt{S}_2, \mathtt{T}_2)$
- else if $\mathtt{S} = \{\mathtt{k}_j : \mathtt{S}_j\ ^{j \in 1..m}\}$ and $\mathtt{T} = \{\mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n}\}$
  - then $\{\mathtt{l}_i\ ^{i \in 1..n}\} \subseteq \{\mathtt{k}_j\ ^{j \in 1..m}\}$
    - $\wedge$ for all $i \in 1..n$ there is some $j \in 1..m$ with $\mathtt{k}_j = \mathtt{l}_i$
      - and $subtype(\mathtt{S}_j, \mathtt{T}_i)$
- else *false*.

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Is our *subtype* function a decision procedure?

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Is our *subtype* function a decision procedure?

Since *subtype* is just an implementation of the algorithmic subtyping rules, we have

1. if $subtype(\mathtt{S}, \mathtt{T}) = true$, then $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$ (hence, by soundness of the algorithmic rules, $\mathtt{S} <: \mathtt{T}$)
2. if $subtype(\mathtt{S}, \mathtt{T}) = false$, then not $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$ (hence, by completeness of the algorithmic rules, not $\mathtt{S} <: \mathtt{T}$)

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Is our *subtype* function a decision procedure?

Since *subtype* is just an implementation of the algorithmic subtyping rules, we have

1. if $subtype(\mathtt{S}, \mathtt{T}) = true$, then $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$ (hence, by soundness of the algorithmic rules, $\mathtt{S} <: \mathtt{T}$)
2. if $subtype(\mathtt{S}, \mathtt{T}) = false$, then not $\blacktriangleright\ \mathtt{S} <: \mathtt{T}$ (hence, by completeness of the algorithmic rules, not $\mathtt{S} <: \mathtt{T}$)

Q: What's missing?

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Is our *subtype* function a decision procedure?

Since *subtype* is just an implementation of the algorithmic subtyping rules, we have

1. if $subtype(\mathtt{S},\mathtt{T}) = true$, then $\mapsto \mathtt{S} <: \mathtt{T}$
   (hence, by soundness of the algorithmic rules, $\mathtt{S} <: \mathtt{T}$)
2. if $subtype(\mathtt{S},\mathtt{T}) = false$, then not $\mapsto \mathtt{S} <: \mathtt{T}$
   (hence, by completeness of the algorithmic rules, not $\mathtt{S} <: \mathtt{T}$)

Q: What's missing?

A: How do we know that *subtype* is a *total* function?

Decision Procedures

Recall: A *decision procedure* for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that $p(u) = true$ iff $u \in R$.

Is our *subtype* function a decision procedure?

Since *subtype* is just an implementation of the algorithmic subtyping rules, we have

1. if $subtype(\mathtt{S},\mathtt{T}) = true$, then $\mapsto \mathtt{S} <: \mathtt{T}$
   (hence, by soundness of the algorithmic rules, $\mathtt{S} <: \mathtt{T}$)
2. if $subtype(\mathtt{S},\mathtt{T}) = false$, then not $\mapsto \mathtt{S} <: \mathtt{T}$
   (hence, by completeness of the algorithmic rules, not $\mathtt{S} <: \mathtt{T}$)

Q: What's missing?

A: How do we know that *subtype* is a *total* function?

Prove it!

Metatheory of Typing

Issue

For the typing relation, we have just one problematic rule to deal with: subsumption.

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

Where is this rule really needed?

Issue

For the typing relation, we have just one problematic rule to deal with: subsumption.

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

Where is this rule really needed?

For applications. E.g., the term

```
(λr:{x:Nat}. r.x) {x=0,y=1}
```

is not typable without using subsumption.

Issue

For the typing relation, we have just one problematic rule to deal with: subsumption.

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

Where is this rule really needed?

For applications. E.g., the term

```
(λr:{x:Nat}. r.x) {x=0,y=1}
```

is not typable without using subsumption.

Where else??

Issue

For the typing relation, we have just one problematic rule to deal with: subsumption.

$$\dfrac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \qquad \text{(T-Sub)}$$

Where is this rule really needed?

For applications. E.g., the term

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

is not typable without using subsumption.

Where else??

Nowhere else! Uses of subsumption to help typecheck applications are the only interesting ones.

Example (T-Abs)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x{:}S_1 \vdash s_2 : S_2} \qquad \dfrac{\vdots}{S_2 <: T_2}}{\Gamma, x{:}S_1 \vdash s_2 : T_2}\,\text{(T-Sub)}}{\Gamma \vdash \lambda x{:}S_1.s_2 : S_1 \to T_2}\,\text{(T-Abs)}$$

Example (T-Abs)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x{:}S_1 \vdash s_2 : S_2} \qquad \dfrac{\vdots}{S_2 <: T_2}}{\Gamma, x{:}S_1 \vdash s_2 : T_2}\,\text{(T-Sub)}}{\Gamma \vdash \lambda x{:}S_1.s_2 : S_1 \to T_2}\,\text{(T-Abs)}$$

becomes

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x{:}S_1 \vdash s_2 : S_2}}{\Gamma \vdash \lambda x{:}S_1.s_2 : S_1 \to S_2}\,\text{(T-Abs)} \qquad \dfrac{\dfrac{}{S_1 <: S_1}\,\text{(S-Refl)} \qquad \dfrac{\vdots}{S_2 <: T_2}}{S_1 \to S_2 <: S_1 \to T_2}\,\text{(S-Arrow)}}{\Gamma \vdash \lambda x{:}S_1.s_2 : S_1 \to T_2}\,\text{(T-Sub)}$$

Example (T-App on the left)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : S_{11} \to S_{12}} \qquad \dfrac{\dfrac{\vdots}{T_{11} <: S_{11}} \qquad \dfrac{\vdots}{S_{12} <: T_{12}}}{S_{11} \to S_{12} <: T_{11} \to T_{12}}\,\text{(S-Arrow)}}{\Gamma \vdash s_1 : T_{11} \to T_{12}}\,\text{(T-Sub)} \qquad \dfrac{\vdots}{\Gamma \vdash s_2 : T_{11}}}{\Gamma \vdash s_1\ s_2 : T_{12}}\,\text{(T-App)}$$

Example (T-App on the left)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : S_{11} \to S_{12}} \qquad \dfrac{\dfrac{\vdots}{T_{11} <: S_{11}} \qquad \dfrac{\vdots}{S_{12} <: T_{12}}}{S_{11} \to S_{12} <: T_{11} \to T_{12}}\,\text{(S-Arrow)}}{\Gamma \vdash s_1 : T_{11} \to T_{12}}\,\text{(T-Sub)} \qquad \dfrac{\vdots}{\Gamma \vdash s_2 : T_{11}}}{\Gamma \vdash s_1\ s_2 : T_{12}}\,\text{(T-App)}$$

becomes

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : S_{11} \to S_{12}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash s_2 : T_{11}} \qquad \dfrac{\vdots}{T_{11} <: S_{11}}}{\Gamma \vdash s_2 : S_{11}}\,\text{(T-Sub)}}{\Gamma \vdash s_1\ s_2 : S_{12}}\,\text{(T-App)} \qquad \dfrac{\vdots}{S_{12} <: T_{12}}}{\Gamma \vdash s_1\ s_2 : T_{12}}\,\text{(T-Sub)}$$

Example (T-App on the right)

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : T_{11} \to T_{12}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash s_2 : T_2} \qquad \dfrac{\vdots}{T_2 <: T_{11}}}{\Gamma \vdash s_2 : T_{11}}\,\text{(T-Sub)}}{\Gamma \vdash s_1\ s_2 : T_{12}}\,\text{(T-App)}$$

Example (T-App on the right)

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : T_{11}\to T_{12}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash s_2 : T_2} \qquad \dfrac{\vdots}{T_2 <: T_{11}}}{\Gamma \vdash s_2 : T_{11}}\;(\text{T-Sub})}{\Gamma \vdash s_1\ s_2 : T_{12}}\;(\text{T-App})$$

becomes

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s_1 : T_{11}\to T_{12}} \qquad \dfrac{\dfrac{\vdots}{T_2 <: T_{11}} \qquad \dfrac{}{T_{12} <: T_{12}}\;(\text{S-Refl})}{T_{11}\to T_{12} <: T_2\to T_{12}}\;(\text{S-Arrow})}{\Gamma \vdash s_1 : T_2\to T_{12}}\;(\text{T-Sub}) \qquad \dfrac{\vdots}{\Gamma \vdash s_2 : T_2}}{\Gamma \vdash s_1\ s_2 : T_{12}}\;(\text{T-App})$$

Example (T-Sub)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s : S} \qquad \dfrac{\vdots}{S <: U}}{\Gamma \vdash s : U}\;(\text{T-Sub}) \qquad \dfrac{\vdots}{U <: T}}{\Gamma \vdash s : T}\;(\text{T-Sub})$$

Example (T-Sub)

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash s : S} \qquad \dfrac{\vdots}{S <: U}}{\Gamma \vdash s : U}\;(\text{T-Sub}) \qquad \dfrac{\vdots}{U <: T}}{\Gamma \vdash s : T}\;(\text{T-Sub})$$

becomes

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash s : S} \qquad \dfrac{\dfrac{\vdots}{S <: U} \qquad \dfrac{\vdots}{U <: T}}{S <: T}\;(\text{S-Trans})}{\Gamma \vdash s : T}\;(\text{T-Sub})$$