# CIS 500

Software Foundations

Fall 2005

21 November

# Administrivia

- Homework 8 will be on website by end of today. Due Wednesday, Novemeber 30.
- Class on Wednesday, but no recitations due to Thanksgiving.
- No office hours on Thursday or Friday.
- View your exams with Cheryl Hickey (Levine 502). (Or Kamila Mauro tomorrow.)
- Regrade requests by Monday. Pick up exams next week.

# Subtyping (Review)

# Subtype relation

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

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

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

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

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

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \qquad \text{(S-Arrow)}$$

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

# Subsumption Rule

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

Other typing rules as in $\lambda_{\rightarrow}$

# Subtyping with Other Features

# Subtyping and Variants

$$\texttt{<}l_i\texttt{:}T_i\ ^{i\in 1..n}\texttt{>} \quad \texttt{<:} \quad \texttt{<}l_i\texttt{:}T_i\ ^{i\in 1..n+k}\texttt{>} \qquad \text{(S-VariantWidth)}$$

$$\frac{\text{for each } \mathtt{i} \quad S_i \texttt{ <: } T_i}{\texttt{<}l_i\texttt{:}S_i\ ^{i\in 1..n}\texttt{>} \quad \texttt{<:} \quad \texttt{<}l_i\texttt{:}T_i\ ^{i\in 1..n}\texttt{>}} \qquad \text{(S-VariantDepth)}$$

$$\frac{\texttt{<}k_j\texttt{:}S_j\ ^{j\in 1..n}\texttt{>} \text{ is a permutation of } \texttt{<}l_i\texttt{:}T_i\ ^{i\in 1..n}\texttt{>}}{\texttt{<}k_j\texttt{:}S_j\ ^{j\in 1..n}\texttt{>} \quad \texttt{<:} \quad \texttt{<}l_i\texttt{:}T_i\ ^{i\in 1..n}\texttt{>}} \qquad \text{(S-VariantPerm)}$$

$$\frac{\Gamma \vdash t_1 \texttt{ : } T_1}{\Gamma \vdash \texttt{<}l_1\texttt{=}t_1\texttt{>} \texttt{ : } \texttt{<}l_1\texttt{:}T_1\texttt{>}} \qquad \text{(T-Variant)}$$

# Subtyping and Lists

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1}{\texttt{List S}_1 \texttt{ <: List T}_1} \qquad \text{(S-List)}$$

I.e., `List` is a covariant type constructor.

# Subtyping references

# Subtyping and References

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Ref S}_1 \texttt{ <: Ref T}_1} \qquad \text{(S-Ref)}$$

I.e., `Ref` is not a covariant (nor a contravariant) type constructor.

Why?

# Subtyping and References

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Ref S}_1 \texttt{ <: Ref T}_1} \qquad \text{(S-Ref)}$$

I.e., `Ref` is not a covariant (nor a contravariant) type constructor.

Why?

- When a reference is read, the context expects a $\texttt{T}_1$, so if $\texttt{S}_1 \texttt{ <: } \texttt{T}_1$ then an $\texttt{S}_1$ is ok.

# Subtyping and References

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Ref S}_1 \texttt{ <: Ref T}_1} \qquad \text{(S-Ref)}$$

I.e., `Ref` is not a covariant (nor a contravariant) type constructor.

Why?

- When a reference is read, the context expects a $\texttt{T}_1$, so if $\texttt{S}_1 \texttt{ <: } \texttt{T}_1$ then an $\texttt{S}_1$ is ok.
- When a reference is written, the context provides a $\texttt{T}_1$ and if the actual type of the reference is $\texttt{Ref S}_1$, someone else may use the $\texttt{T}_1$ as an $\texttt{S}_1$. So we need $\texttt{T}_1 \texttt{ <: } \texttt{S}_1$.

# Subtyping and Arrays

Similarly...

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-Array)}$$

# Subtyping and Arrays

Similarly...

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1 \qquad \texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-ARRAY)}$$

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1}{\texttt{Array S}_1 \texttt{ <: Array T}_1} \qquad \text{(S-ARRAYJAVA)}$$

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

# References again

Observation: a value of type `Ref T` can be used in two different ways: as a source for values of type `T` and as a sink for values of type `T`.

# References again

Observation: a value of type `Ref T` 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 `Ref T` into three parts:

- `Source T`: reference cell with "read cabability"
- `Sink T`: reference cell with "write cabability"
- `Ref T`: cell with both capabilities

# Modified Typing Rules

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Source}\ T_{11}}{\Gamma \mid \Sigma \vdash\ !t_1 : T_{11}} \qquad \text{(T-Deref)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Sink}\ T_{11} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \texttt{Unit}} \qquad \text{(T-Assign)}$$

# Subtyping rules

$$\frac{\texttt{S}_1 \texttt{ <: } \texttt{T}_1}{\texttt{Source S}_1 \texttt{ <: Source T}_1} \qquad \text{(S-Source)}$$

$$\frac{\texttt{T}_1 \texttt{ <: } \texttt{S}_1}{\texttt{Sink S}_1 \texttt{ <: Sink T}_1} \qquad \text{(S-Sink)}$$

$$\texttt{Ref T}_1 \texttt{ <: Source T}_1 \qquad \text{(S-RefSource)}$$

$$\texttt{Ref T}_1 \texttt{ <: Sink T}_1 \qquad \text{(S-RefSink)}$$

# 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.

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

If we are given some $\Gamma$ 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 ($\Gamma$ 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" $\Gamma$ 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)

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

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!

   (Hence, 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

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

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 `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

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

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

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

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

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

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \qquad \text{(S-Arrow)}$$

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

# Issues

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

1. S-Rcd-Width, S-Rcd-Depth, and S-Rcd-Perm overlap with each other
2. S-Refl and S-Trans overlap with everything

# Step 1: simplify record subtyping

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

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

# 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: `S <: S` can be derived for every type without using S-Refl.

# Even simpler subtype relation

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

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

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \qquad \text{(S-Arrow)}$$

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

# Step 3: Get rid of transitivity

Observation: S-Trans is unnecessary.

Lemma: If `S <: T` can be derived, then there is a derivation that does not use S-Trans.

# "Algorithmic" subtype relation

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

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

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

# Soundness and completeness

Theorem: `S <: T` iff $\vdash$ `S <: T`.

Proof: ...

Terminology:

- The algorithmic presentation of subtyping is sound with respect to the original if $\vdash$ `S <: T` implies `S <: T`.

  (Everything validated by the algorithm is actually true.)

- The algorithmic presentation of subtyping is complete with respect to the original if `S <: T` implies $\vdash$ `S <: T`.

  (Everything true is validated by the algorithm.)

# Subtyping Algorithm (pseudo-code)

The algorithmic rules can be translated directly into code:

$$
\begin{array}{lll}
\mathit{subtype}(\mathtt{S},\mathtt{T}) & = & \text{if } \mathtt{T} = \mathtt{Top}, \text{ then } \mathit{true} \\
& & \text{else if } \mathtt{S} = \mathtt{S_1}\to\mathtt{S_2} \text{ and } \mathtt{T} = \mathtt{T_1}\to\mathtt{T_2} \\
& & \quad \text{then } \mathit{subtype}(\mathtt{T_1},\mathtt{S_1}) \ \wedge \ \mathit{subtype}(\mathtt{S_2},\mathtt{T_2}) \\
& & \text{else if } \mathtt{S} = \{\mathtt{k_j}:\mathtt{S_j}\ ^{j\in 1..m}\} \text{ and } \mathtt{T} = \{\mathtt{l_i}:\mathtt{T_i}\ ^{i\in 1..n}\} \\
& & \quad \text{then} \quad \{\mathtt{l_i}\ ^{i\in 1..n}\} \subseteq \{\mathtt{k_j}\ ^{j\in 1..m}\} \\
& & \qquad \wedge \text{ for all } \mathtt{i} \text{ there is some } \mathtt{j} \in \mathtt{1..m} \text{ with } \mathtt{k_j} = \mathtt{l_i} \\
& & \qquad\quad \text{and } \mathit{subtype}(\mathtt{S_j},\mathtt{T_i}) \\
& & \text{else } \mathit{false}.
\end{array}
$$

# Decision Procedures

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

# Decision Procedures

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

Is our *subtype* function a decision procedure?

# Decision Procedures

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

Is our *subtype* function a decision procedure?

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

1. if *subtype*(`S`, `T`) = ***true***, then $\vdash$ `S <: T`

   (hence, by soundness of the algorithmic rules, `S <: T`)

2. if *subtype*(`S`, `T`) = ***false***, then not $\vdash$ `S <: T`

   (hence, by completeness of the algorithmic rules, not `S <: T`)

# Decision Procedures

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

Is our *subtype* function a decision procedure?

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

1. if *subtype*(`S`, `T`) = ***true***, then $\vdash$ `S <: T`

   (hence, by soundness of the algorithmic rules, `S <: T`)

2. if *subtype*(`S`, `T`) = ***false***, then not $\vdash$ `S <: T`

   (hence, by completeness of the algorithmic rules, not `S <: T`)

Q: What's missing?

# Decision Procedures

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

Is our *subtype* function a decision procedure?

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

1. if *subtype*(`S`, `T`) = ***true***, then $\vdash$ `S <: T`

   (hence, by soundness of the algorithmic rules, `S <: T`)

2. if *subtype*(`S`, `T`) = ***false***, then not $\vdash$ `S <: T`

   (hence, by completeness of the algorithmic rules, not `S <: T`)

Q: What's missing?

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

# Decision Procedures

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

Is our *subtype* function a decision procedure?

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

1. if *subtype*(`S`, `T`) = ***true***, then $\vdash$ `S <: T`

   (hence, by soundness of the algorithmic rules, `S <: T`)

2. if *subtype*(`S`, `T`) = ***false***, then not $\vdash$ `S <: T`

   (hence, by completeness of the algorithmic rules, not `S <: 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.

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

Where is this rule really needed?

# Issue

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

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

Where is this rule really needed?

For applications. E.g., the term

```
(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.

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

Where is this rule really needed?

For applications. E.g., the term

```
(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.

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

Where is this rule really needed?

For applications. E.g., the term

```
(λr:{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

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

# Example

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

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

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

# Example

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

becomes

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

# Example

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

# Example

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

becomes

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

# Example

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

# Example

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

becomes

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