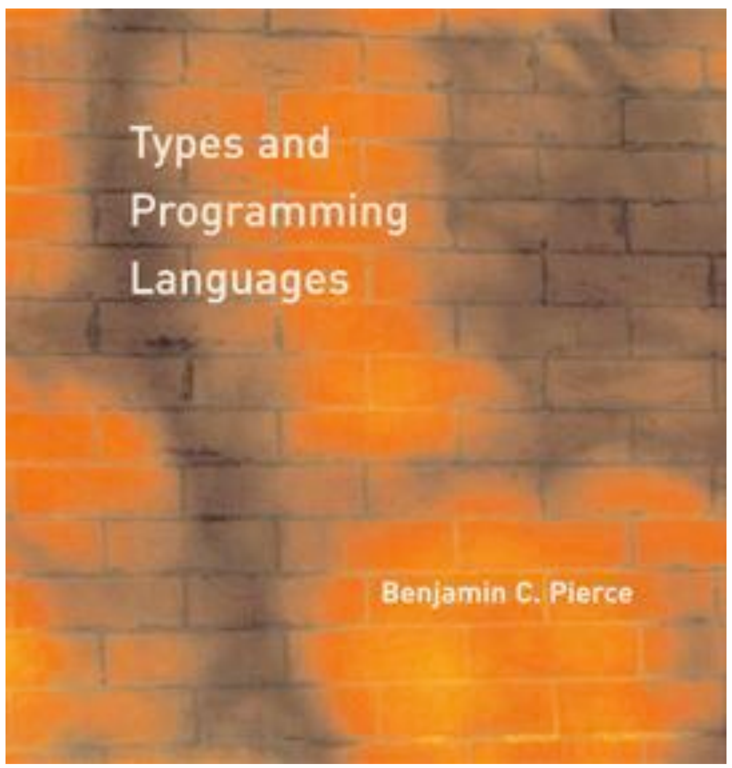
Programming Languages
Fall 2013



Lecture 8: More on Simply-Typed Lambda Calculus:
substitution lemma, preservation, erasure and type inference

Prof. Liang Huang

huang@qc.cs.cuny.edu

Typing rules

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Bool} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T} \qquad \Gamma \vdash \texttt{t}_3 : \texttt{T}}{\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

$$\frac{\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \lambda \texttt{x}{:}\texttt{T}_1\texttt{.t}_2 : \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(T-Abs)}$$

$$\frac{\texttt{x}{:}\texttt{T} \in \Gamma}{\Gamma \vdash \texttt{x} : \texttt{T}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{T}_{12} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{T}_{12}} \qquad \text{(T-App)}$$

Typing derivations

Exercise 9.2.2: Show (by drawing derivation trees) that the following terms have the indicated types:

1. $\texttt{f:Bool}{\rightarrow}\texttt{Bool} \vdash \texttt{f (if false then true else false)} : \texttt{Bool}$
2. $\texttt{f:Bool}{\rightarrow}\texttt{Bool} \vdash \lambda\texttt{x:Bool. f (if x then false else x)} : \texttt{Bool}{\rightarrow}\texttt{Bool}$

The two typing relations

Question: What is the relation between these two statements?

1. `t : T`
2. $\vdash$ `t : T`

The two typing relations

Question: What is the relation between these two statements?

1. $\texttt{t : T}$
2. $\vdash \texttt{t : T}$

First answer: These two relations are completely different things.

- We are dealing with several different small programming languages, *each with its own typing relation* (between terms in that language and types in that language)
- For the simple language of numbers and booleans, typing is a *binary* relation between terms and types ($\texttt{t : T}$).
- For $\lambda_{\rightarrow}$, typing is a *ternary* relation between contexts, terms, and types ($\Gamma \vdash \texttt{t : T}$).

  (When the context is empty — because the term has no free variables — we often write $\vdash \texttt{t : T}$ to mean $\emptyset \vdash \texttt{t : T}$.)

Conservative extension

Second answer: The typing relation for $\lambda_{\rightarrow}$ *conservatively extends* the one for the simple language of numbers and booleans.

- Write "language 1" for the language of numbers and booleans and "language 2" for the simply typed lambda-calculus with base types `Nat` and `Bool`.
- The terms of language 2 include all the terms of language 1; similarly typing rules.
- Write $\texttt{t} :_1 \texttt{T}$ for the typing relation of language 1.
- Write $\Gamma \vdash \texttt{t} :_2 \texttt{T}$ for the typing relation of language 2.
- *Theorem:* Language 2 conservatively extends language 1: If `t` is a term of language 1 (involving only booleans, conditions, numbers, and numeric operators) and `T` is a type of language 1 (either `Bool` or `Nat`), then $\texttt{t} :_1 \texttt{T}$ iff $\emptyset \vdash \texttt{t} :_2 \texttt{T}$.

Preservation (and Weaking, Permutation, Substitution)

Review: Proving progress

Let's quickly review the steps in the proof of the progress theorem:

- inversion lemma for typing relation
- canonical forms lemma
- progress theorem

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x:T}_1\texttt{.t}_2 : \texttt{R}$, then

Inversion

Lemma:

1. If $\Gamma \vdash \mathtt{true} : \mathtt{R}$, then $\mathtt{R} = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : \mathtt{R}$, then $\mathtt{R} = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{R}$, then $\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool}$ and $\Gamma \vdash \mathtt{t}_2, \mathtt{t}_3 : \mathtt{R}$.
4. If $\Gamma \vdash \mathtt{x} : \mathtt{R}$, then $\mathtt{x}{:}\mathtt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{R}$, then $\mathtt{R} = \mathtt{T}_1 \rightarrow \mathtt{R}_2$ for some $\mathtt{R}_2$ with $\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{R}_2$.
6. If $\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{R}$, then

Inversion

Lemma:

1. If $\Gamma \vdash \texttt{true} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
2. If $\Gamma \vdash \texttt{false} : \texttt{R}$, then $\texttt{R} = \texttt{Bool}$.
3. If $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x:T}_1\texttt{.t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \rightarrow \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.
6. If $\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{R}$, then there is some type $\texttt{T}_{11}$ such that $\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \rightarrow \texttt{R}$ and $\Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}$.

Canonical Forms

Lemma:

Canonical Forms

Lemma:

1. If $\mathtt{v}$ is a value of type $\mathtt{Bool}$, then $\mathtt{v}$ is either $\mathtt{true}$ or $\mathtt{false}$.
2. If $\mathtt{v}$ is a value of type $\mathtt{T_1}{\to}\mathtt{T_2}$, then $\mathtt{v}$ has the form $\lambda\mathtt{x}\mathtt{:}\mathtt{T_1}\mathtt{.}\mathtt{t_2}$.

Progress

Theorem: Suppose $\mathtt{t}$ is a closed, well-typed term (that is, $\vdash \mathtt{t} : \mathtt{T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Steps of proof:

- Weakening
- Permutation
- Substitution preserves types
- Reduction preserves types (i.e., preservation)

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{x} \notin \mathit{dom}(\Gamma)$, then $\Gamma, \texttt{x}{:}\texttt{S} \vdash \texttt{t} : \texttt{T}$.

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{x} \notin \mathit{dom}(\Gamma)$, then $\Gamma, \texttt{x}{:}\texttt{S} \vdash \texttt{t} : \texttt{T}$.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and Δ is a permutation of Γ, then $\Delta \vdash \texttt{t} : \texttt{T}$.

Weakening and Permutation

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin \mathit{dom}(\Gamma)$, then $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$.

Moreover, the latter derivation has the same depth as the former.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ, then $\Delta \vdash \mathtt{t} : \mathtt{T}$.

Moreover, the latter derivation has the same depth as the former.

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction

Preservation

Theorem: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{T}$.

Proof: By induction on typing derivations.

Which case is the hard one??

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on typing derivations.

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on typing derivations.

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on typing derivations.

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $\mathtt{t}_1 = \lambda \mathtt{x}{:}\mathtt{T}_{11}.\ \mathtt{t}_{12}$

$\mathtt{t}_2$ a value $\mathtt{v}_2$

$\mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on typing derivations.

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $\mathtt{t}_1 = \lambda \mathtt{x}{:}\mathtt{T}_{11}.\ \mathtt{t}_{12}$

$\mathtt{t}_2$ a value $\mathtt{v}_2$

$\mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

Uh oh.

Preservation

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on typing derivations.

Case T-APP: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $\mathtt{t}_1 = \lambda \mathtt{x}:\mathtt{T}_{11}.\ \mathtt{t}_{12}$

$\mathtt{t}_2$ a value $\mathtt{v}_2$

$\mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

Uh oh. What do we need to know to make this case go through??

The “Substitution Lemma”

Lemma: If $\Gamma, \texttt{x}{:}\texttt{S} \vdash \texttt{t} : \texttt{T}$ and $\Gamma \vdash \texttt{s} : \texttt{S}$, then $\Gamma \vdash [\texttt{x} \mapsto \texttt{s}]\texttt{t} : \texttt{T}$.

I.e., “Types are preserved under substitition.”

The "Substitution Lemma"

Lemma: If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the *depth* of a derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

The "Substitution Lemma"

Lemma: If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the *depth* of a derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

The "Substitution Lemma"

Lemma: If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the *depth* of a derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

Case T-APP:

$$\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$$
$$\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t}_1 : \mathtt{T}_2 \to \mathtt{T}_1$$
$$\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t}_2 : \mathtt{T}_2$$
$$\mathtt{T} = \mathtt{T}_1$$

By the induction hypothesis, $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1 : \mathtt{T}_2 \to \mathtt{T}_1$ and $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2 : \mathtt{T}_2$. By T-APP, $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2 : \mathtt{T}$, i.e., $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) : \mathtt{T}$.

The "Substitution Lemma"

Lemma: If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the *depth* of a derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

Case T-VAR: $\mathtt{t} = \mathtt{z}$
with $\mathtt{z}{:}\mathtt{T} \in (\Gamma, \mathtt{x}{:}\mathtt{S})$

There are two sub-cases to consider, depending on whether $\mathtt{z}$ is $\mathtt{x}$ or another variable. If $\mathtt{z} = \mathtt{x}$, then $[\mathtt{x} \mapsto \mathtt{s}]\mathtt{z} = \mathtt{s}$. The required result is then $\Gamma \vdash \mathtt{s} : \mathtt{S}$, which is among the assumptions of the lemma. Otherwise, $[\mathtt{x} \mapsto \mathtt{s}]\mathtt{z} = \mathtt{z}$, and the desired result is immediate.

The "Substitution Lemma"

Lemma: If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the *depth* of a derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

Case T-ABS:
$$\mathtt{t} = \lambda \mathtt{y}{:}\mathtt{T}_2.\mathtt{t}_1 \qquad \mathtt{T} = \mathtt{T}_2 \to \mathtt{T}_1$$
$$\Gamma, \mathtt{x}{:}\mathtt{S}, \mathtt{y}{:}\mathtt{T}_2 \vdash \mathtt{t}_1 : \mathtt{T}_1$$

By our conventions on choice of bound variable names, we may assume $\mathtt{x} \neq \mathtt{y}$ and $\mathtt{y} \notin FV(\mathtt{s})$. Using *permutation* on the given subderivation, we obtain $\Gamma, \mathtt{y}{:}\mathtt{T}_2, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t}_1 : \mathtt{T}_1$. Using *weakening* on the other given derivation ($\Gamma \vdash \mathtt{s} : \mathtt{S}$), we obtain $\Gamma, \mathtt{y}{:}\mathtt{T}_2 \vdash \mathtt{s} : \mathtt{S}$. Now, by the induction hypothesis, $\Gamma, \mathtt{y}{:}\mathtt{T}_2 \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1 : \mathtt{T}_1$. By T-ABS, $\Gamma \vdash \lambda \mathtt{y}{:}\mathtt{T}_2.\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1 : \mathtt{T}_2 \to \mathtt{T}_1$, i.e. (by the definition of substitution), $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\lambda \mathtt{y}{:}\mathtt{T}_2.\ \mathtt{t}_1 : \mathtt{T}_2 \to \mathtt{T}_1$.

Erasure and Typability

Erasure

We can transform terms in $\lambda_{\rightarrow}$ to terms of the untyped lambda-calculus simply by erasing type annotations on lambda-abstractions.

$$\begin{array}{lcl} \mathit{erase}(\mathtt{x}) & = & \mathtt{x} \\ \mathit{erase}(\lambda\mathtt{x}\mathtt{:}\mathtt{T}_1\mathtt{.}\ \mathtt{t}_2) & = & \lambda\mathtt{x}.\ \mathit{erase}(\mathtt{t}_2) \\ \mathit{erase}(\mathtt{t}_1\ \mathtt{t}_2) & = & \mathit{erase}(\mathtt{t}_1)\ \mathit{erase}(\mathtt{t}_2) \end{array}$$

Typability

Conversely, an untyped λ-term $\mathtt{m}$ is said to be *typable* if there is some term $\mathtt{t}$ in the simply typed lambda-calculus, some type $\mathtt{T}$, and some context Γ such that $\mathit{erase}(\mathtt{t}) = \mathtt{m}$ and $\Gamma \vdash \mathtt{t} : \mathtt{T}$.

This process is called *type reconstruction* or *type inference*.

Typability

Conversely, an untyped λ-term $\mathtt{m}$ is said to be *typable* if there is some term $\mathtt{t}$ in the simply typed lambda-calculus, some type $\mathtt{T}$, and some context Γ such that $\mathit{erase}(\mathtt{t}) = \mathtt{m}$ and $\Gamma \vdash \mathtt{t} : \mathtt{T}$.

This process is called *type reconstruction* or *type inference*.

Example: Is the term

$$\lambda\mathtt{x.\ x\ x}$$

typable?

More About Bound Variables

Substitution

Our definition of evaluation is based on the "substitution" of values for free variables within terms.

$$(\lambda \mathtt{x}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-AppAbs)}$$

But what is substitution, exactly? How do we define it?

Substitution

For example, what does

$$(\lambda\texttt{x. x } (\lambda\texttt{y. x y})) \; (\lambda\texttt{x. x y x})$$

reduce to?

Note that this example is not a "complete program" — the whole term is not closed. We are mostly interested in the reduction behavior of closed terms, but reduction of open terms is also important in some contexts:

- program optimization
- alternative reduction strategies such as "full beta-reduction"

Formalizing Substitution

Consider the following definition of substitution:

$$
\begin{aligned}
&[x \mapsto s]x = s \\
&[x \mapsto s]y = y && \text{if } x \neq y \\
&[x \mapsto s](\lambda y.t_1) = \lambda y.\ ([x \mapsto s]t_1) \\
&[x \mapsto s](t_1\ t_2) = ([x \mapsto s]t_1)([x \mapsto s]t_2)
\end{aligned}
$$

What is wrong with this definition?

Formalizing Substitution

Consider the following definition of substitution:

$$
\begin{aligned}
&[x \mapsto s]x = s \\
&[x \mapsto s]y = y && \text{if } x \neq y \\
&[x \mapsto s](\lambda y.t_1) = \lambda y.\ ([x \mapsto s]t_1) \\
&[x \mapsto s](t_1\ t_2) = ([x \mapsto s]t_1)([x \mapsto s]t_2)
\end{aligned}
$$

What is wrong with this definition?

It substitutes for free and *bound* variables!

$$[x \mapsto y](\lambda x.\ x) = \lambda x.y$$

This is not what we want!

Substitution, take two

$$
\begin{array}{lr}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) = \lambda\mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{x}.\mathtt{t}_1) = \lambda\mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

Substitution, take two

$$
\begin{aligned}
&[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} && \\
&[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} && \text{if } \mathtt{x} \neq \mathtt{y} \\
&[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) = \lambda\mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) && \text{if } \mathtt{x} \neq \mathtt{y} \\
&[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{x}.\mathtt{t}_1) = \lambda\mathtt{x}.\ \mathtt{t}_1 && \\
&[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &&
\end{aligned}
$$

What is wrong with this definition?

It suffers from *variable capture*!

$$[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{y}.\mathtt{x}) = \lambda\mathtt{x}.\ \mathtt{x}$$

This is also not what we want.

Substitution, take three

$$
\begin{aligned}
&[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} \\
&[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} && \text{if } \mathtt{x} \neq \mathtt{y} \\
&[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) = \lambda\mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) && \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin FV(\mathtt{s}) \\
&[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{x}.\mathtt{t}_1) = \lambda\mathtt{x}.\ \mathtt{t}_1 \\
&[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2)
\end{aligned}
$$

What is wrong with this definition?

Substitution, take three

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) = \lambda\mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin FV(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{x}.\mathtt{t}_1) = \lambda\mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

Now substition is a *partial function!*

E.g., $[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{y}.\mathtt{x})$ is undefined.

But we want an result for every substitution.

Bound variable names shouldn't matter

It's annoying that that the "spelling" of bound variable names is causing trouble with our definition of substitution.

Intuition tells us that there shouldn't be a difference between the functions $\lambda\texttt{x.x}$ and $\lambda\texttt{y.y}$. Both of these functions do exactly the same thing.

Because they differ only in the names of their bound variables, we'd like to think that these *are* the same function.

We call such terms *alpha-equivalent*.

Alpha-equivalence classes

In fact, we can create equivalence classes of terms that differ only in the names of bound variables.

When working with the lambda calculus, it is convenient to think about these *equivalence classes*, instead of raw terms.

For example, when we write $\lambda \mathtt{x.x}$ we mean not just this term, but the class of terms that includes $\lambda \mathtt{y.y}$ and $\lambda \mathtt{z.z}$.

We can now freely choose a different *representative* from a term's alpha-equivalence class, whenever we need to, to avoid getting stuck.

Substitution, for alpha-equivalence classes

Now consider substitution as an operation over *alpha-equivalence classes* of terms.

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) = \lambda\mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin FV(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{x}.\mathtt{t}_1) = \lambda\mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

Examples:

- $[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{y}.\mathtt{x})$ must give the same result as $[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{z}.\mathtt{x})$. We know the latter is $\lambda\mathtt{z}.\mathtt{y}$, so that is what we will use for the former.
- $[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{x}.\mathtt{z})$ must give the same result as $[\mathtt{x} \mapsto \mathtt{y}](\lambda\mathtt{w}.\mathtt{z})$. We know the latter is $\lambda\mathtt{w}.\mathtt{z}$ so that is what we use for the former.

Review

So what does

$$(\lambda\texttt{x. x } (\lambda\texttt{y. x y}))\ (\lambda\texttt{x. x y x})$$

reduce to?