

## Comp 411 Notes - Fri, Feb 4, 2005

### Substitutivity

Scribe: Moez A. Abdel-Gawad  
(moez@cs.rice.edu)

This lecture discusses the important property of *substitutivity* in typed lambda calculus, where a variable in an expression, or a subexpression of that expression, can be replaced by another expression of the same type as the variable or subexpression, while the type of the whole original expression remains unchanged.

## 1 Typed Lambda Calculus

Assuming:

$$\begin{aligned} b &\in \mathbb{B} \supseteq \{bool, int, string, \dots\} \\ x &\in X \\ e &::= x \mid \lambda x.e \mid e e \\ t &::= b \mid t \rightarrow t \\ \Gamma &::= [] \mid \Gamma, (x, t) \quad (\text{or, } \Gamma ::= [] \mid \Gamma, (x:t)) \quad (x \notin dom(\Gamma)) \end{aligned}$$

consider the typing rules:

$$\frac{x : t \in \Gamma}{\Gamma \vdash x : t} \quad \frac{\Gamma, x : t_1 \vdash e : t_2}{\Gamma \vdash \lambda x.e : t_1 \rightarrow t_2} \quad \frac{\Gamma \vdash e_1 : t_1 \rightarrow t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1 e_2 : t_2}$$

## 2 Basic Properties of Typed Lambda Calculus

Let's try to type  $\lambda x.\lambda x.x$ :

Without the requirement of unique variable names ( $\alpha$ -renaming) in the domain of  $\Gamma$ , *both* of the following type derivations are possible:

$$\frac{\frac{\frac{x : \mathbf{string} \in \Gamma, (x : string), (x : int)}{\Gamma, (x : string), (x : int) \vdash x : \mathbf{string}}}{\Gamma, x : string \vdash \lambda x.x : int \rightarrow \mathbf{string}}}{\Gamma \vdash \lambda x.\lambda x.x : string \rightarrow (int \rightarrow \mathbf{string})}}$$

$$\frac{\frac{\frac{x : \mathbf{int} \in \Gamma, (x : string), (x : int)}{\Gamma, (x : string), (x : int) \vdash x : \mathbf{int}}}{\Gamma, x : string \vdash \lambda x.x : int \rightarrow \mathbf{int}}}{\Gamma \vdash \lambda x.\lambda x.x : string \rightarrow (int \rightarrow \mathbf{int})}}$$

This is clearly undesirable, because any assumptions we make about  $x$  (the bound occurrence) would depend on the type of  $x$ , but having multiple types allows no assumptions to be made, deeming the type system useless.

Note, however, that if we used de Bruijn's notation for lambda calculus' syntax, we'd get the following typing rules (assuming  $\Gamma' ::= [] \mid \Gamma', t$ ), where we would not have ambiguities for the  $x$ 's in  $\Gamma$ :

$$\frac{}{\Gamma', t_1 \vdash \#0 : t_1} \qquad \frac{\Gamma' \vdash \#n : t_2}{\Gamma', t_1 \vdash \#n + 1 : t_2}$$

$$\frac{\Gamma', t_1 \vdash e' : t_2}{\Gamma' \vdash \lambda.e' : t_1 \rightarrow t_2} \qquad \frac{\Gamma' \vdash e'_1 : t_1 \rightarrow t_2 \quad \Gamma' \vdash e'_2 : t_1}{\Gamma' \vdash e'_1 e'_2 : t_2}$$

The reason de Bruijn notation avoids ambiguities in  $\Gamma$  is that the binding occurrences of variables are uniquely identified (by integers) in  $\Gamma$  (or the insertion *order* in  $\Gamma$  uniquely identifies them). In de Bruijn notation, no two binding occurrences are referred to using the same identifier/name. In addition, all bound occurrences of variables refer to these unique identifiers (integers).

This shows the necessity of the  $\alpha$ -renaming in the standard typing rules of Typed Lambda Calculus. With  $\alpha$ -renaming, we get the following derivation:

$$\frac{\mathbf{y} : \mathit{int} \in \Gamma, (x : \mathit{string}), (\mathbf{y} : \mathit{int})}{\Gamma, (x : \mathit{string}), (\mathbf{y} : \mathit{int}) \vdash y : \mathit{int} \quad (\alpha\text{-renaming})}}{\frac{\Gamma, x : \mathit{string} \vdash \lambda x.x : \mathit{int} \rightarrow \mathbf{int}}{\Gamma \vdash \lambda x.\lambda x.x : \mathit{string} \rightarrow (\mathit{int} \rightarrow \mathbf{int})}}$$

However, we still have a “problem” in the LC type system. Note, for example, that the  $\lambda x.x$ , in the expression above, for any specific type  $t$ , could have had the type  $t \rightarrow t$ . We, thus, lost unicity of types. A typical solution to such situation is to ask for explicit type annotations, by the programmer (for example,  $e ::= x \mid \lambda x:\underline{t}.e \mid e e$ ).

## 2.1 Unicity of Types

Unicity of types states that:

$$\forall e.\forall \Gamma.\forall t.\forall t'. \Gamma \vdash e : t \wedge \Gamma \vdash t' \implies t = t'$$

Another solution that restores unicity, in a sense, is to use *type schemas* instead of types. A type schema can contain *type variables*, and a type variable can stand for an element of a (possibly infinite) set of types (e.g.,  $\alpha \rightarrow \alpha$  is a type schema, where  $\alpha$  is a type variable that stands for all possible types in  $t$ ).

### 3 Substitutivity

However, the most important property of Typed Lambda Calculus is *substitutivity*, where we can replace a variable in an expression with a term of the same type, and we end with the same type for the whole expression unchanged.

**Lemma 3.1** *Substitution Lemma*

$$\frac{\Gamma, x : t_1 \vdash e_1 : t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1[x := e_2] : t_2}$$

Proof of Lemma (Sketch of it):

By induction on the derivation of the type of  $e_1$ :

*Case*  $e_1 = y$  (Some variable  $y$ )

$$\begin{array}{l} y[x := e_2] \text{ if } y = x \\ y[x := e_2] \text{ if } y \neq x \end{array}$$

*Case*  $e_1 = \lambda x.e$

First we have:

$$\frac{\Gamma, x : t, y : t_3 \vdash e : t_4}{\Gamma, x : t_1 \vdash \lambda y.e : t_3 \rightarrow t_4}$$

(after  $\alpha$ -renaming & choosing  $y$  that does not occur in  $e_2$  (for the next substitution to work))

$$\frac{\Gamma \vdash e_2 : t_1}{\Gamma \vdash (\lambda y.e)[x := e_2] : t_3 \rightarrow t_4}$$

Since  $(\lambda y.e)[x := e_2] \equiv \lambda y.(e[x := e_2])$  then we can push the substitution inwards:

$$\frac{\Gamma \vdash e_2 : t_1}{\Gamma, y : t_3 \vdash e[x := e_2] : t_4}$$

To further proceed in the proof, we need the following lemma

**Lemma 3.2** *Exchange Lemma*

$$\frac{\Gamma_1, x : t_1, y : t_2, \Gamma_2 \vdash e : t}{\Gamma_1, y : t_2, x : t_1, \Gamma_2 \vdash e : t}$$

and that both have the same derivation size.

**HW:** Prove the two lemmas above.