CS611 Lecture 23 **Substitution and Term Equivalence** 22 October 2003

Scribe: Mohamed Elhawary Lecturer: Radu Rugina

1 Introduction

Last lecture, while showing the evaluation of the factorial function using the call by value combinator Y_{CBV}, we mentioned that we use the equal sign to equivalent expression. In this lecture we will formally define the concept of term equivalence. We will also give a precise definition for substitution, we we used many times fo far, without formally defining it.

2 Substitution

When performing substitution, we must consider the following two problems:

- Bound Variables: Which variable should be replaced? Consider the following example:

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

  So, we should only replace the free variables.

- Variable Capture: We have to make sure that free variables before the substitution, remain free after the substitution, i.e. they are not going to be captured by abstractions. Consider the following example:

$$\lambda\ y.x[x \to y] = \lambda\ y'.y$$

  So, it should not result in $\lambda\ y.y$ which is the identity function. Instead, the expression is the constant function.

Now, the substitution can be defined inductively as:

- Variable:

$$\begin{aligned} x\ [x \to e] &= \ e \\ y\ [x \to e] &= \ y \quad , \qquad \text{if } y \neq x \end{aligned}$$

- Application

$$e_0\ e_1\ [x \to e]\ =\ (e_0\ [x \to e])\ (e_1\ [x \to e])$$

- Abstraction

$$\begin{array}{lcll} (\lambda\ x.e_0)\ [x \to e] & = & \lambda\ x.e_0 & \\ (\lambda\ y.e_0)\ [x \to e] & = & \lambda\ y.e_0\ [x \to e] & \text{if } y \neq x \wedge y \notin FV(e) \\ (\lambda\ y.e_0)\ [x \to e] & = & \lambda\ y'.(e_0\ [y \to y'])[x \to e] & \text{if } y \neq x \wedge y \in FV(e) \wedge \\ & & & y' \notin (FV(e) \cup FV(e_0) \cup \{x\}) \end{array}$$

  Where free variable are defined as follows:

$$\begin{array}{ll} FV(x) & = \{x\} \\ FV(e_0\ e_1) & = FV(e_0) \cup FV(e_1) \\ FV(\lambda\ x.e) & = FV(e) - \{x\} \end{array}$$

  Let's see a simple example:

$$\begin{array}{rcl} (\lambda\ y.(\lambda\ x\ .\ y\ x)\ x)\ [x \to y\ x] & = & \lambda\ y'.(((\lambda\ x\ .\ y\ x)\ x\ [y \to y'])[x \to yx]) \\ & = & \lambda\ y'.((\lambda\ x\ .\ y'\ x)\ x\ [x \to y\ x]) \\ & = & \lambda\ y'.((\lambda\ x.\ y'\ x)\ y\ x) \end{array}$$

3 Equivalence

When we studied the Y_{CBV} combinator, we had:

$$Y_{CBV}F \to_1^* F\ (\lambda\, y.\, Y_{CBV}\ Fy)$$

Then we said that the expression $\lambda\, y.\, Y_{CBV}\ Fy$ and $Y_{CBV}F$ are equivalent, to conclude that $Y_{CBV}F$ and $F(Y_{CBV}F)$ are equivalent. We now formally define this equivalence.

3.1 Reductions

- η -reduction: we would like to say that two expressions are equivalent if they return the same result given the same arguments. η -reduction is defined as follow:

  $$(\lambda\, x.e\ x) \to_\eta e$$

  when you apply both of them on the same argument, you will get the same result.

- α -reduction: It allows us to rename the variables of an expression. We saw it before in substitution. This is formalized as:

  $$(\lambda\, x.e) \to_\alpha (\lambda\, y.e\ [x \to y])$$

- β -reduction:

  $$(\lambda\, x.e)\ e' \to_\beta (e[x \to e'])$$

3.2 Compatible Closure and Equivalence Relation

We can write any of the above reduction as a relation R, which is a relation on λ terms. We will expand this relation into compatible closure, which is one step evaluation relation.

- Compatible Closure $\to_R$

  $$\begin{aligned} (x,y) \in R \Longrightarrow &\quad x \to_R y \\ x \to_R y \Longrightarrow &\quad z\ x \to_R z\ y \\ x \to_R y \Longrightarrow &\quad x\ z \to_R y\ z \\ x \to_R y \Longrightarrow &\quad (\lambda\, z.x) \to_R (\lambda\, z.y) \end{aligned}$$

- Transitive and Reflexive Closure $\to_R^*$

  $$\begin{aligned} x \to_R y \Longrightarrow &\quad x \to_R^* y \\ x \to_R^* x & \\ x \to_R^* y \wedge y \to_R^* z \Longrightarrow &\quad x \to_R^* y \end{aligned}$$

- Equation Relation $=_R$

  $$\begin{aligned} x \to_R^* y \Longrightarrow &\quad x =_R y \\ x =_R y \Longrightarrow &\quad y =_R x \\ x =_R y \wedge y =_R z \Longrightarrow &\quad x =_R z \end{aligned}$$

  With these definitions, we are able now to use $=_\beta$, $=_\eta$, $=_\alpha$, $\to_\beta^*$, $\to_\eta^*$ and $\to_\alpha^*$. We can also combine two relations and define those closures for the combination of relations. For example, $\beta\eta = \beta \cup \eta$ then:

  $$x \to_{\beta\eta} y \Longleftrightarrow (x \to_\beta y) \vee (x \to_\eta y)$$

If we return to the example of the Y_{CBV} combinator, we can show the equivalence of $Y\ F = F(Y\ F)$ as follows:

$$\begin{aligned} Y_{CBV}F \to_\beta^* F(\lambda\, y.\, Y_{CBV}F\ y) \to_\eta &\ F(Y_{CBV}\ F) \\ Y_{CBV}F =_{\beta\eta} &\ F(Y_{CBV}\ F) \end{aligned}$$

4 Normal Forms and Evaluation Order

4.1 Normal Forms

A normal form is an expression which cannot be further reduced. In particular, we can have β-nf, η-nf, or $\beta\eta$-nf which means that there is no applicable β-reduction, η-reduction, or $\beta\eta$-reduction.

Does an expression always has a normal form? consider the following example:

$$\begin{aligned} \mathit{self} &= \lambda\, x.\, xx \\ \mathit{omega} &= \mathit{self\ self} \rightarrow_\beta \mathit{self\ self} \end{aligned}$$

We can see that this expression does not have a β-nf.

4.2 Evaluation Order

Since there may be multiple β-reductions that can be applied in any order, we need to decide which ones to pick in such cases. We already discussed two possibilities, CBV and CBN. Today, we present three more:

- *Normal Order.* Evaluate the outermost, leftmost redex (reducible expression). Equivalent to CBN, but we additionally reduce inside the abstraction bodies. It can find a normal form if there exists one. Rules of normal order:

$$\frac{}{(\lambda\, x.\, e_0)e_1 \rightarrow_1 e_0\ [x \mapsto e_1]} \qquad \frac{e_0 \rightarrow_1 e_0'}{\lambda\, x.\, e_0' \rightarrow_1 \lambda\, x.\, e_0'}$$

$$\frac{na \rightarrow_1 e_0'}{na\ e_1 \rightarrow_1 e_0'\ e_1} \qquad \frac{e_1 \rightarrow_1 e_1'}{nanf\ e_1 \rightarrow_1 nanf\ e_1'}$$

  where na and $nanf$ means non abstraction and non abstraction normal form, i.e it is used when the expression is not falling in the category of the first rule. And, they are recursively defined as follows:

$$\begin{aligned} na &= x \mid e_0\ e_1 \\ nanf &= x \mid nanf\ nf \\ nf &= \lambda\, x.\, nf \mid nanf \end{aligned}$$

- *Applicative Order.* Evaluates the innermost, leftmost redex. Equivalent to CBV with reducing inside the abstraction. It can not always find a normal form. Rules are left as exercise.

- *Full β-reduction.* This is the relation $\rightarrow_R$ defined above, for $R = \beta$. It allows reductions to be applied in any order. The choice and the evaluation is non-deterministic. It can evaluate whatever can be evaluated by any of the other evaluation orders, if we make the appropriate choices during the evaluation. It can to find nf if if there is one.