cs611 Lecture 23 **Eta reduction and Evaluation Orders** 10/23/2002

Scribe: Sigmund Cherem & Oren Kurland Lecturer: Radu Rugina

1 Introduction

When we defined the β-reduction, we had to be careful doing a substitution. We had to change the names of free variables and consider the problem of variable capture. For instance,

$$(\lambda x.\,(\lambda y.\,xy)y)[y \mapsto xy] =_? (\lambda x'.\,(\lambda y.\,x'y)[y \mapsto yx]$$

In order to do this steps, without lose of formality, we need to introduce new types of reductions and equivalence between expressions.

2 α-Equivalence

By introducing α-equivalence we look forward to solve the above problem. The main purpose is to be able to rename variables, for instance:

$$\lambda x.\,x =_\alpha \lambda y.\,y$$

To formalize this we define the α-reduction as follows:

$$\lambda x.\,e \rightarrow_\alpha \lambda x'.\,(e[x \mapsto x'])$$

Where $x' \notin FV(e)$.

We want to be able to use this relation as an equivalence. This has many consequences, for instance, after applying an α-reduction, we can always apply another α-reduction to get back the original expression. All this features can be very useful in many applications, for example, to prove the equivalence of different expressions.

Now, the α-equivalence is defined inductively as follows:

$$\frac{}{\lambda x.\,e =_\alpha \lambda x'.\,(e[x \mapsto x'])}\ x' \notin FV(e)$$

The following two rules are known as the *compatible closure*:

$$\frac{e_0 =_\alpha e_0' \quad e_1 =_\alpha e_1'}{e_0\ e_1 =_\alpha e_0'\ e_1'} \qquad \frac{e_0 =_\alpha e_0'}{\lambda x.\,e_0 =_\alpha \lambda x.\,e_0'}$$

And the following three define (respectively) the *reflexive, symmetric and transitive closure*:

$$\frac{}{e_0 =_\alpha e_0} \qquad \frac{e_1 =_\alpha e_0}{e_0 =_\alpha e_1} \qquad \frac{e_0 =_\alpha e_1 \quad e_1 =_\alpha e_2}{e_0 =_\alpha e_2}$$

But α-reduction is very constraint. Until now we have β and α reductions. With them we can reduce somehow the structure of the expressions, and change names of variables whenever we need. To be able to effectively reduce more the expressions, we need another type of reduction.

3 η-reduction

Suppose we have to different expressions f and g, such that $f\ x \rightarrow_1^* g\ x$. We want to be able to say that $f = g$. For instance, when we studied the Y combinator we had:

$$Y_{CBV}\ F \rightarrow_1^* F(\lambda y.\,Y_{CBV}\ F\ y)$$

Then we said that $\lambda y.\,Y_{CBV}\ F\ y = Y_{CBV}\ F$ and reduced the whole expression to $F(Y_{CBV}\ F)$. This last reduction is called a η-reduction, and it's defined as follows:

$$\lambda x.\,e\ x \rightarrow_\eta e$$

As we did with the α-reduction, we would like to define compatible closures and equivalent relations for this reduction. In order to do this, we first introduce a general definition that can be applied to any reduction defined until now.

4 Generalization of compatible closure and equivalence relation

Let R be a relation (R can stand for β, α or η in our case). We define the compatible closure "$\rightarrow_R$" as follows:

$$\begin{array}{lll}
(x, y) \in R & \Rightarrow & x \rightarrow_R y \\
(x, y) \in R & \Rightarrow & (\lambda z.\, x) \rightarrow_R (\lambda z.\, y) \\
(x, y) \in R & \Rightarrow & x\ z \rightarrow_R y\ z \\
(x, y) \in R & \Rightarrow & z\ x \rightarrow_R z\ y
\end{array}$$

To extend it to a symmetric, reflexive and transitive closure $\rightarrow_R^*$ we need to add the following:

$$\begin{array}{lll}
x \rightarrow_R y & \Rightarrow & x \rightarrow_R^* y \\
 & & x \rightarrow_R^* x \\
x \rightarrow_R^* y \wedge y \rightarrow_R^* z & \Rightarrow & x \rightarrow_R^* z
\end{array}$$

Then, we can define an equivalence relation as follows:

$$\begin{array}{lll}
x \rightarrow_R y & \Rightarrow & x =_R y \\
x =_R y & \Rightarrow & y =_R x \\
x =_R y \wedge y =_R z & \Rightarrow & x =_R z
\end{array}$$

Note that we needed to include again the second and third clauses to be able to preserve symmetry and transitivity on the equivalence relation.

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

$$x \rightarrow_{\beta\eta} y \Leftrightarrow x \rightarrow_\beta y \vee x \rightarrow_\eta y$$

Now that we defined this relation, we can complete our previous example of the Y combinator, and properly say that:

$$Y_{CBV}\ F =_{\beta\eta} F(Y_{CBV}\ F)$$

5 Normal Forms and Evaluation Order

Until now we use the term *redex* to denote any reducible expression. Now we have to refer to them as $\beta - redex$ or $\eta - redex$ depending on the applicable reduction.

If an expression does not have any *redex* available to reduce, we say we have a *normal form*. We name $\beta - nf$ a normal for of β-reductions, this means, an expression where there is no $\beta - redex$. Similarly we define $\eta - nf$.

In general, we don't have a guarantee to always have a normal form on any expression. For example, the following expression doesn't have a $\beta - nf$:

$$omega = (\lambda x.\, x\ x)(\lambda x.\, x\ x)$$

You may notice that $omega \rightarrow_\beta omega$. In a similar way, the Y combinator doesn't have a $\eta - nf$.

In general, we are interested in reducing expressions to a normal form. Depending in the order we apply our rules, we are going to be able to get to a different final state. Here we present four evaluation orders:

5.1 CBV/CBN

We are only able to apply β-reductions, in any of those approaches: CBV or CBN. In such sense, we can get expressions that are not reducibly, but we would like to reduce them further. For instance:

$$\begin{array}{lll}
inc\ c_1 & = & (\lambda n.\, \lambda s.\, \lambda z.\, s(n\ s\ z))(\lambda s.\, \lambda z.\, s\ z) \\
 & \rightarrow_1 & \lambda s.\, \lambda z.\, s((\lambda s.\, \lambda z.\, s\ z)\ s\ z) \\
 & \not\rightarrow_1 & \lambda s.\, \lambda z.\, s(s\ z) \\
 & \rightarrow_1 & c_2
\end{array}$$

The third step can't be done because we have an abstraction and until now, we are not allowed to apply any reduction in this type of expressions.

5.2 Normal order

We would like to evaluate the outermost, leftmost *redex*. In such sense when we have an expression like $inc(inc\ c_0)$ it will behave like CBN, substituting first, reducing later.

When we have an abstraction $\lambda x.\, inc\ c_1$, we will evaluate the body and try to reduce it as much as possible. Formally this is defined as follows:

$$\begin{array}{lcll} na & ::= & x \mid e_0\, e_1 & \text{Not abstraction} \\ nanf & ::= & x \mid nanf\ nf & \text{Not abstraction, normal form} \\ nf & ::= & \lambda x.\, nf \mid nanf & \text{Normal form} \end{array}$$

$$\frac{e \rightarrow_1^{NO} e'}{\lambda x.\, e \rightarrow_1^{NO} \lambda x.\, e'}$$

$$\frac{}{(\lambda x.\, e_0)e_1 \rightarrow_1^{NO} e_0[x \mapsto e_1]}\ \beta - red$$

$$\frac{na \rightarrow_1^{NO} e_0}{na\ e_1 \rightarrow_1^{NO} e_0\ e_1} \qquad \frac{e_1 \rightarrow_1^{NO} e_1'}{nanf\ e_1 \rightarrow_1^{NO} nanf\ e_1'}$$

Note that we reduce na (expressions that aren't abstractions) until we get an abstraction or a normal form. In case it is an abstraction, we apply a β-reduction. In case is a not abstraction, but normal form ($nanf$), we apply the last rule in order to reduce the rest of the expression.

For example

$$((\lambda x.\, \lambda y.\, 1)2)3 \rightarrow_1 (\lambda y.\, 1)3 \rightarrow_1 1$$

where the first reduction was done applying the third rule, and the second reduction was done applying the second (β-reduction) rule.

5.3 Applicative order

We would like to evaluate the innermost, leftmost *redex*. In such sense when we have an expression like $inc(inc\ c_0)$ it will behave like CBV, reducing the arguments before doing an application.

Later, as we had in Normal Order, when we have an abstraction $\lambda x.\, inc\ c_1$, we will evaluate the body and try to reduce it as much as possible. The formal definition is very similar to Normal order.

5.4 Full β-reduction

Intuitively is defined as the combination of the last two. This evaluation order represents a closure of β-reduction.

$$\frac{e \rightarrow e'}{\lambda x.\, e \rightarrow \lambda x.\, e'}$$

$$\frac{}{(\lambda x.\, e_0)e_1 \rightarrow e_0[x \mapsto e_1]}\ \beta - red$$

$$\frac{e_0 \rightarrow e_0'}{e_0\ e_1 \rightarrow e_0'\ e_1} \qquad \frac{e_1 \rightarrow e_1'}{e_0\ e_1 \rightarrow e_0\ e_1'}$$

This evaluation order is non deterministic. At each step we will have to choose which rule we want to apply. On the other hand, any other evaluation order will get as far as this one. If an expression has a normal form, this Full β-reduction can choose the appropriate sequence of rules to get into such normal form.

For instance, the applicative order will never get a normal form while reducing an expression like $(\lambda y.\, 1)$ *omega*, but normal order will. In other cases it will happen the opposite way. Only full β reduction will be able to reduce both situations at the same time.

The main problem with this evaluation order is that a bad selection of the rules to apply will lead us to have the same problems we had with the others.