1 Lambda calculus evaluation

There are many different evaluation strategies for the $\lambda$-calculus. The most permissive is full $\beta$ reduction, which allows any redex—i.e., any expression of the form $(\lambda x . e_1) e_2$—to step to $e_1[e_2/x]$ at any time. It is defined formally by the following small-step operational semantics rules:

\[
\begin{align*}
  e_1 & \rightarrow e'_1 \\
  e_1 e_2 & \rightarrow e'_1 e_2 \\
  e_2 & \rightarrow e'_2 \\
  \lambda x . e_1 & \rightarrow \lambda x . e'_1 \\
  (\lambda x . e_1) e_2 & \rightarrow e_1[e_2/x]
\end{align*}
\]

The call by value (CBV) strategy enforces a more restrictive strategy: it only allows an application to reduce after its argument has been reduced to a value (i.e., a $\lambda$-abstraction) and does not allow evaluation under a $\lambda$. It is described by the following small-step operational semantics rules (here we show a left-to-right version of CBV):

\[
\begin{align*}
  e_1 & \rightarrow e'_1 \\
  e_1 e_2 & \rightarrow e'_1 e_2 \\
  e_2 & \rightarrow e'_2 \\
  v_1 e_2 & \rightarrow v_1 e'_2 \\
  (\lambda x . e_1) v_2 & \rightarrow e_1[v_2/x]
\end{align*}
\]

Finally, the call by name (CBN) strategy allows an application to reduce even when its argument is not a value but does not allow evaluation under a $\lambda$. It is described by the following small-step operational semantics rules:

\[
\begin{align*}
  e_1 & \rightarrow e'_1 \\
  e_1 e_2 & \rightarrow e'_1 e_2 \\
  (\lambda x . e_1) e_2 & \rightarrow e_1[e_2/x]
\end{align*}
\]

2 Confluence

It is not hard to see that the full $\beta$ reduction strategy is non-deterministic. This raises an interesting question: does the choices made during the evaluation of an expression affect the final result? The answer turns out to be no: full $\beta$ reduction is confluent in the following sense:

**Theorem (Confluence).** If $e \rightarrow^* e_1$ and $e \rightarrow^* e_2$ then there exists $e'$ such that $e_1 \rightarrow^* e'$ and $e_2 \rightarrow^* e'$.

Confluence can be depicted graphically as follows:

![Confluence Diagram]

Confluence is often also called the Church-Rosser property.
3 Substitution

Each of the evaluation relations for λ-calculus has a β defined in terms of a substitution operation on expressions. Because the expressions involved in the substitution may share some variable names (and because we are working up to α-equivalence) the definition of this operation is slightly subtle and defining it precisely turns out to be tricker than might first appear.

As a first attempt, consider an obvious (but incorrect) definition of the substitution operator. Here we are substituting \( e \) for \( x \) in some other expression:

\[
\begin{align*}
y{e/x} & = \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases} \\
(e_1 e_2){e/x} & = (e_1{e/x})(e_2{e/x}) \\
(\lambda y.e_1){e/x} & = \lambda y.(e_1{e/x}) \quad \text{where } y \neq x
\end{align*}
\]

The intuitive idea is that the last rule relies on α-equivalence to “rewrite” abstractions that use \( x \) so they do not conflict. Unfortunately, this definition produces the wrong results when we substitute an expression with free variables under a λ. For example,

\[
(\lambda y.x){y/x} = (\lambda y.y)
\]

To fix this problem, we need to revise our definition so that when we substitute under a λ we do not accidentally bind variables in the expression we are substituting. The following definition correctly implements capture-avoiding substitution:

\[
\begin{align*}
y{e/x} & = \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases} \\
(e_1 e_2){e/x} & = (e_1{e/x})(e_2{e/x}) \\
(\lambda y.e_1){e/x} & = \lambda y.(e_1{e/x}) \quad \text{where } y \neq x \text{ and } y \notin \text{fv}(e)
\end{align*}
\]

Note that in the case for λ-abstractions, we require that the bound variable \( y \) be different from the variable \( x \) we are substituting for and that \( y \) not appear in the free variables of \( e \), the expression we are substituting. Because we work up to α-equivalence, we can always pick \( y \) to satisfy these side conditions. For example, to calculate \( (\lambda z.x \ z){(w \ y \ z)/x} \) we first rewrite \( \lambda z.x \ z \) to \( \lambda u.x \ u \) and then apply the substitution, obtaining \( \lambda u.(w \ y \ z) \ u \) as the result.