CSci 4651 Fall 2008

Handout 1: Call-by-name vs. call-by-value λ-calculus (a brief overview)

Notations: Beta-reduction for the call-by-name λ-calculus is marked by β_N, beta-reduction for the call-by-value λ-calculus is marked by β_V.

Common definitions. Both call-by-name and call-by-value λ-calculi consist of terms defined as follows:

$$M \to x \mid n \mid \lambda x.M \mid M_1 M_2,$$

where x denotes a variable, n denotes a number. $\lambda x.M$ is called a lambda-abstraction and denotes a function with the body M and the parameter x. $M_1 M_2$ denotes an application, for instance $(\lambda x.x)5$ denotes an application of an identity function (a function that just returns the parameter without doing anything to it) to a number 5. Even though arithmetic operations, such as $+, -$, etc. are not included in the definition of terms, I will use these operations in examples.

Free and bound variables, substitution, and alpha-equivalence are defined the same way for both calculi. See the textbook for more details.

Both calculi evaluate λ-terms via β-reduction. However, this reduction is defined differently for the two calculi.

Call-by-name λ-calculus. The basic β-reduction in the call-by-name calculus is defined as follows:

$$(\lambda x.M)N \xrightarrow[\text{simple}]{\beta_N} \left[\frac{N}{x}\right] M,$$

where $[\frac{N}{x}]M$ denotes the result of replacing all free occurrences of x in M by N.

In order to reason about program transformations we extend this relation so that β-reduction can be performed anywhere in a λ-term (i.e. in any subterm). The complete β-reduction is defined as follows:

$$\begin{aligned} M \xrightarrow{\beta_N} N \quad & \text{if} \quad M \xrightarrow[\text{simple}]{\beta_N} N, \text{ or} \\ & \text{if} \quad M = \lambda x.M_1, N = \lambda x.N_1, \text{ and } M_1 \xrightarrow{\beta_N} N_1, \text{ or} \\ & \text{if} \quad M = M_1 M_2, N = N_1 N_2, \text{ and } M_1 \xrightarrow{\beta_N} N_1,\ M_2 = N_2, \text{ or} \\ & \text{if} \quad M = M_1 M_2, N = N_1 N_2, \text{ and } M_2 \xrightarrow{\beta_N} N_2,\ M_1 = N_1 \end{aligned}$$

Call-by-value λ-calculus. The basic β-reduction in the call-by-value calculus requires that the argument is evaluated (i.e. is a value) before it is substituted into the function:

$$(\lambda x.M)V \xrightarrow[\text{simple}]{\beta_V} \left[\frac{V}{x}\right] M,$$

where a value is either a variable, or a number, or a lambda-abstraction (i.e. a function):

$$V \to x \mid n \mid \lambda x.M$$

For instance, $y, 5, \lambda x.x, \lambda x.2+3$ are all values, but $3+4, x*y$, and $(\lambda x.x)5$ are not.

The complete β-reduction is defined exactly as in the call-by-name λ-calculus, but uses $\xrightarrow[\text{simple}]{\beta_V}$ instead of $\xrightarrow[\text{simple}]{\beta_N}$ in the base case.

$$\begin{aligned} M \xrightarrow{\beta_V} N \quad & \text{if} \quad M \xrightarrow[\text{simple}]{\beta_V} N, \text{ or} \\ & \text{if} \quad M = \lambda x.M_1, N = \lambda x.N_1, \text{ and } M_1 \xrightarrow{\beta_V} N_1, \text{ or} \\ & \text{if} \quad M = M_1 M_2, N = N_1 N_2, \text{ and } M_1 \xrightarrow{\beta_V} N_1,\ M_2 = N_2, \text{ or} \\ & \text{if} \quad M = M_1 M_2, N = N_1 N_2, \text{ and } M_2 \xrightarrow{\beta_V} N_2,\ M_1 = N_1 \end{aligned}$$

Difference in the order of evaluation. The call-by-name λ-calculus substitutes a parameter into the body of the function without evaluating it, the call-by-value λ-calculus requires that the parameter is evaluated to a value before it gets substituted. For instance,

$$\begin{aligned} \underline{(\lambda x.x)((\lambda y.y)(\lambda z.z))} &\xrightarrow{\beta_N} \underline{(\lambda y.y)(\lambda z.z)} \xrightarrow{\beta_N} \lambda z.z \\ (\lambda x.x)\underline{((\lambda y.y)(\lambda z.z))} &\xrightarrow{\beta_V} \underline{(\lambda x.x)(\lambda z.z)} \xrightarrow{\beta_V} \lambda z.z, \end{aligned}$$

The underlining shows which part of the term gets evaluated.

Note that the result of the evaluation is the same in both calculi, but the order of evaluation is different.

In some cases the behavior of the same term may be different in the call-by-value and the call-by-name λ-calculi. Before we look at an example, consider the following term:

$$(\lambda x.xx)(\lambda x.xx) \xrightarrow{\beta_N} (\lambda x.xx)(\lambda x.xx) \xrightarrow{\beta_N} \dots$$

As you can see, this terms evaluates to itself in the call-by-name λ-calculus.

Exercise: Check that the above term evaluates to itself in the call-by-value λ-calculus as well.

Since the term always evaluates to itself, the evaluation will never stop. Such terms are called *diverging* terms. This particular one is often denoted by Ω (capital greek letter omega).

The following term has different behavior in the two calculi:

$$\begin{aligned} \underline{(\lambda x.5)\Omega} &\xrightarrow{\beta_N} 5 \\ (\lambda x.5)\underline{\Omega} &\xrightarrow{\beta_V} (\lambda x.5)\underline{\Omega} \xrightarrow{\beta_V} \dots \end{aligned}$$

In the call-by-name evaluation of $(\lambda x.5)((\lambda x.xx)(\lambda x.xx))$ the diverging term gets replaced for all free occurrences of x in the body of $\lambda x.5$. Since the body of this function is just 5 (i.e. it doesn't have any free occurrences of x), the whole term immediately evaluates to 5.

In the call-by-value evaluation we have to evaluate the parameter first, since, as it is, the parameter is not a value. An attempt to evaluate the parameter

$(\lambda x.xx)(\lambda x.xx)$ leads to the same term. Therefore the evaluation of the parameter will never stop, and the evaluation of the whole term $(\lambda x.5)((\lambda x.xx)(\lambda x.xx))$ goes on forever.

Relation between the two λ-calculi. The following results about the differences between the call-by-name and the call-by-value evaluation have been proven:

1. If both the call-by-value and the call-by-name evaluation of a term M stop, then the results of such evaluation are the same.
2. If a term M reaches a normal form (a term that cannot be reduced any further) in the call-by-value λ-calculus, then it reaches a normal form in the call-by-name λ-calculus (but not the other way around, as we have seen in the previous example).

Another way of formulating the first property is: "If a term M reaches a normal form in both the call-by-name and the call-by-value λ-calculi, then these normal forms are exactly the same."

Notice that Scheme is evaluated according to the call-by-value λ-calculus, not the call-by-name.

Confluence. Both λ-calculi have the confluence property. I will formulate this property for the call-by-name λ-calculus, you can rewrite it for the call-by-value λ-calculus by replacing $\xrightarrow{\beta_N}$ by $\xrightarrow{\beta_V}$. Notice a slight difference in definition of confluence here and in the textbook.

Confluence of the call-by-name λ-calculus. If M evaluates to M_1 in $n_1 \geq 0$ steps of $\xrightarrow{\beta_N}$ and M evaluates to M_2 in $n_2 \geq 0$ steps of $\xrightarrow{\beta_N}$, then there exists a term N such that M_1 evaluates to N in $n_3 \geq 0$ steps of $\xrightarrow{\beta_N}$ and M_2 evaluates to N in $n_4 \geq 0$ steps of $\xrightarrow{\beta_N}$.

Example: consider the following two different β-reductions from the same term M:

$$M = (\lambda x.\underline{2+3})((\lambda y.y)(\lambda z.z)) \xrightarrow{\beta_N} (\lambda x.5)((\lambda y.y)(\lambda z.z)) = M_1,$$
$$M = (\lambda x.2+3)\underline{((\lambda y.y)(\lambda z.z))} \xrightarrow{\beta_N} (\lambda x.2+3)(\lambda z.z) = M_2,$$

We can reduce M_1 and M_2 to the same term N as follows:

$$M_1 = (\lambda x.5)\underline{((\lambda y.y)(\lambda z.z))} \xrightarrow{\beta_N} (\lambda x.5)(\lambda z.z) = N$$
$$M_2 = (\lambda x.\underline{2+3})(\lambda z.z) \xrightarrow{\beta_N} (\lambda x.5)(\lambda z.z) = N.$$

As in the previous example, the part of the term that gets reduced is underlined.

Efficiency vs. safety. We have already shown that the call-by-name λ-calculus is "safer": a program might diverge (i.e. go into an infinite loop) in the call-by-value calculus when it reaches a normal form in the call-by-name.

However, the call-by-value calculus avoids unnecessary repetition of work. Consider the following example:

$$(\lambda x.x*x)(2+3) \xrightarrow{\beta_N} (2+3)*(2+3) \xrightarrow{\beta_N} 5*(2+3) \xrightarrow{\beta_N} 5*5 \xrightarrow{\beta_N} 25.$$
$$(\lambda x.x*x)(2+3) \xrightarrow{\beta_V} (\lambda x.x*x)5 \xrightarrow{\beta_V} 5*5 \xrightarrow{\beta_V} 25.$$

There are two occurrences of x in $\lambda x.x * x$. If we evaluate $2 + 3$ before the substitution (as in call-by-value), then we only need to evaluate it once. If we substitute $2 + 3$ without evaluating it first (as in call-by-name), then we need to evaluate both copies of it later, so we end up doing more computation. Therefore call-by-value is generally a more efficient evaluation model.

Exercise: there are cases, however, when call-by-value does more work than call-by-name. What are these cases? Why is this not a significant issue?