Call-by-name, Call-by-value, and Effects

Alex Kavvos

The substitution lemma says that if $\vdash e : \tau$ and $x : \tau \vdash u : \sigma$ then $\vdash u[e/x] : \sigma$. In other words we may replace a free variable $x : \tau$ with any term of that type. Thus, **variables stand for terms**. This evaluation strategy—the only one we have used until now—is known as **call-by-name (CBN)**.

This approach to free variables comes from mathematics. For example, if we define $f(x) \stackrel{\text{def}}{=} x + x$ then we have

$$f(1+1) = (1+1) + (1+1) = 2 + (1+1) = 2 + 2 = 4$$

However, there would be a slight advantage in the number of evaluation steps if we simplified the function argument to a value first. In that case we would have the much shorter computation

$$f(1+1) = f(2) = 2 + 2 = 4$$

The way to enforce this is to ask that **variables stand for values**, so that a function argument is evaluated before being substituted in the function body. This evaluation strategy is known as **call-by-value (CBV)**.

Of course we know that the order of evaluation does not matter in mathematical or purely functional settings, as either choice leads to the same result. However, it becomes very important when our programs have **effects**.

Call-by-name is strongly associated with purely functional languages with lazy evaluation. A highly optimized form of it known as call-by-need is used in Haskell. Almost all non-pure languages, i.e. languages with effects, use call-by-value. This includes C, Java, Scala, JS, OCaml, Scheme, and so on. (The exception is ALGOL 60.)

1 The call-by-value λ-calculus

The **call-by-value λ-calculus** is a version of the STLC in which all substitutions replace a variable with a value. As a consequence, the changes only affect constructs that use substitution, i.e. sum types and function types.

The changes do not affect the statics of the STLC, as typing remains the same. However, they affect the dynamics, which now behave as follows. D-Inl, D-Inr, and D-App-2 are new; all changes to previous rules are in red.

$$\textsc{Val-Inl}\ \frac{v\ \mathsf{val}}{\mathsf{inl}(v)\ \mathsf{val}} \qquad \textsc{Val-Inr}\ \frac{v\ \mathsf{val}}{\mathsf{inr}(v)\ \mathsf{val}} \qquad \textsc{Val-Lam}\ \frac{}{\lambda x : \tau.\, e\ \mathsf{val}} \qquad \textsc{D-Let}\ \frac{v\ \mathsf{val}}{\mathsf{let}(v; x.\, e_2) \longmapsto e_2[v/x]}$$

$$\textsc{D-Inl}\ \frac{e \longmapsto e'}{\mathsf{inl}(e) \longmapsto \mathsf{inl}(e')} \qquad \textsc{D-Inr}\ \frac{e \longmapsto e'}{\mathsf{inr}(e) \longmapsto \mathsf{inr}(e')} \qquad \textsc{D-Case-Inl}\ \frac{v\ \mathsf{val}}{\mathsf{case}(\mathsf{inl}(v); x.\, e_1; y.\, e_2) \longmapsto e_1[v/x]}$$

$$\textsc{D-Case-Inr}\ \frac{v\ \mathsf{val}}{\mathsf{case}(\mathsf{inr}(v); x.\, e_1; y.\, e_2) \longmapsto e_2[v/y]} \qquad \textsc{D-Case-1}\ \frac{e \longmapsto e'}{\mathsf{case}(e; x.\, e_1; y.\, e_2) \longmapsto \mathsf{case}(e'; x.\, e_1; y.\, e_2)}$$

$$\textsc{D-App-1}\ \frac{e_1 \longmapsto e_1'}{e_1(e_2) \longmapsto e_1'(e_2)} \qquad \textsc{D-App-2}\ \frac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{e_1(e_2) \longmapsto e_1(e_2')} \qquad \textsc{D-Beta}\ \frac{v\ \mathsf{val}}{(\lambda x : \tau.\, e)(v) \longmapsto e[v/x]}$$

Progress and preservation are also true for the CBV λ-calculus.

2 Examples

The calculational example given above can be reproduced in λ-calculus as follows. In the CBN STLC we have

$$\begin{aligned} & \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; \mathsf{num}[1]))(\mathsf{plus}(\mathsf{num}[1]; \mathsf{num}[1]))} \\ \longmapsto_{\mathrm{n}} & \mathsf{plus}(\underline{\mathsf{plus}(\mathsf{num}[1]; \mathsf{num}[1])}; \mathsf{num}[1]) \\ \longmapsto_{\mathrm{n}} & \underline{\mathsf{plus}(\mathsf{num}[2]; \mathsf{num}[1])} \\ \longmapsto_{\mathrm{n}} & \mathsf{num}[3] \end{aligned}$$

However, in the CBV STLC we have

$$\begin{aligned} & \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; \mathsf{num}[1]))(\mathsf{plus}(\mathsf{num}[1]; \mathsf{num}[1]))} \\ \longmapsto_{\mathrm{v}} & \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; \mathsf{num}[1]))(\mathsf{num}[2])} \\ \longmapsto_{\mathrm{v}} & \underline{\mathsf{plus}(\mathsf{num}[2]; \mathsf{num}[1])} \\ \longmapsto_{\mathrm{v}} & \mathsf{num}[3] \end{aligned}$$

The result is the same, but the process is different!

3 Effects

The chasm between CBN and CBV widens in the presence of side-effects. Consider a **printing** primitive:

$$\textsc{Print}\ \frac{s \in \Sigma^* \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{print}(s; e) : \tau}$$

The dynamics are now given by a relation $e \overset{s}{\longmapsto} e'$, read as "$e$ prints s and steps to e'". Of course all rules need to be modified to thread s around—we write ε for the empty string:

$$\text{D-P-Print}\ \frac{}{\mathsf{print}(s; e) \overset{s}{\longmapsto} e} \qquad \text{D-P-Beta}\ \frac{[v\ \mathsf{val}]}{(\lambda x : \tau.\, e)(v) \overset{\varepsilon}{\longmapsto} e[v/x]} \qquad \text{D-P-App-1}\ \frac{e_1 \overset{s}{\longmapsto} e_1'}{e_1(e_2) \overset{s}{\longmapsto} e_1'(e_2)}$$

and so on...

The bracketed premise in D-P-Beta is included in CBV, but not in CBN.

Then a strange issue arises if we consider the term $\vdash (\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; x))(\mathsf{print}(\mathrm{hi}; \mathsf{num}[1])) : \mathsf{Num}$.

In CBV:

$$\begin{aligned} \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; x))(\mathsf{print}(\mathrm{hi}; \mathsf{num}[1]))} & \overset{\mathrm{hi}}{\longmapsto}_{\mathrm{v}} \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; x))(\mathsf{num}[1])} \\ & \overset{\varepsilon}{\longmapsto}_{\mathrm{v}} \mathsf{plus}(\mathsf{num}[1]; \mathsf{num}[1]) \\ & \overset{\varepsilon}{\longmapsto}_{\mathrm{v}} \mathsf{num}[2] \end{aligned}$$

In contrast, in CBN:

$$\begin{aligned} \underline{(\lambda x : \mathsf{Num}.\, \mathsf{plus}(x; x))(\mathsf{print}(\mathrm{hi}; \mathsf{num}[1]))} & \overset{\varepsilon}{\longmapsto}_{\mathrm{n}} \mathsf{plus}(\underline{\mathsf{print}(\mathrm{hi}; \mathsf{num}[1])}; \mathsf{print}(\mathrm{hi}; \mathsf{num}[1])) \\ & \overset{\mathrm{hi}}{\longmapsto}_{\mathrm{n}} \mathsf{plus}(\mathsf{num}[1]; \underline{\mathsf{print}(\mathrm{hi}; \mathsf{num}[1])}) \\ & \overset{\mathrm{hi}}{\longmapsto}_{\mathrm{n}} \mathsf{plus}(\mathsf{num}[1]; \mathsf{num}[1]) \\ & \overset{\varepsilon}{\longmapsto}_{\mathrm{n}} \mathsf{num}[2] \end{aligned}$$

In CBN the program prints "hi" twice, whereas in CBV the same program only prints it once!

That is an **observable** difference that distinguishes the two evaluation strategies.

Without side effects CBN and CBV are observationally identical evaluation strategies (but not if we count the number of evaluation steps).