

Solutions for Exercise Sheet 6

Exercise 1: Call-by-name reduction

a. The first line of call-by-name reduction in Handout 7, paragraph 3, reads:

$$\text{and } (\text{not } (\text{null } l)) ((\text{car } l) > 0) \longrightarrow_{\beta}^* \text{if } (\text{not } (\text{null } l)) ((\text{car } l) > 0) \text{ false}$$

Write down each of the β -reduction steps involved in this reduction. For each step, notate the redex being reduced in square brackets and verify that the reduction context is a valid call-by-name context (according to the grammar given in paragraph 7).

$$\begin{aligned} \text{and } (\text{not } (\text{null } l)) ((\text{car } l) > 0) &= [(\lambda p. \lambda q. \text{if } p \text{ false}) (\text{not } (\text{null } l))] ((\text{car } l) > 0) \\ &\longrightarrow_{\beta} [(\lambda q. \text{if } (\text{not } (\text{null } l)) q \text{ false}) ((\text{car } l) > 0)] & (1) \\ &\longrightarrow_{\beta} \text{if } (\text{not } (\text{null } l)) ((\text{car } l) > 0) \text{ false} & (2) \end{aligned}$$

Step (1) is valid in CBN because $[]$ is a reduction context (by BETA) and so $[] ((\text{car } l) > 0)$ is a reduction context (by GO LEFT).

Step (2) is valid in CBN because $[]$ is a reduction context (by BETA).

b. In the same block, there is allusion to a reduction sequence with the following effect:

$$\text{not } (\text{null } l) \longrightarrow_{\beta, \delta}^* \text{true}$$

Expand this out into individual reduction steps and verify that the reduction contexts are valid call-by-name contexts.

Recall that the function *not* is defined as $\lambda p. \text{if } p \text{ false true}$.

$$\begin{aligned} \text{not } (\text{null } l) &= [(\lambda p. \text{if } p \text{ false true}) (\text{null } l)] \\ &\longrightarrow_{\beta} \text{if } [(\text{null } l)] \text{ false true} & (1) \\ &\longrightarrow_{\delta} [\text{if false false true}] & (2) \\ &\longrightarrow_{\delta} \text{true} & (3) \end{aligned}$$

Step (1) is valid because $[]$ is a reduction context (by BETA).

Step (2) is valid because $[]$ is a reduction context (by DELTA), and so (if $[]$) is a reduction context (by COND), and so (if $[]$ false) is a reduction context by GO LEFT, and so (if $[]$ false true) is a reduction context by GO LEFT.

Step (3) is valid because $[]$ is a reduction context (by DELTA).

Exercise 2: Call-by-value reduction

The function *or* is defined as follows:

$$\text{or} = \lambda p. \lambda q. \text{if } p \text{ true } q$$

a. Using the values $x = 4$ and $n = 2$, evaluate the following term under call by value reduction:

$$\text{or } (n = 0) (x/n = 2)$$

At each step, annotate your choice of redex in square brackets and verify that the reduction context is a valid call-by-value context (as per the grammar in paragraph 9).

$$\begin{aligned}
\text{or } (n = 0) (x/n = 2) &= (\lambda p. \lambda q. \text{if } p \text{ true } q) [(n = 0)] (x/n = 2) \\
&\longrightarrow_{\delta} [(\lambda p. \lambda q. \text{if } p \text{ true } q) \text{false}] (x/n = 2) & (1) \\
&\longrightarrow_{\beta} (\lambda q. \text{if } \text{false true } q) [(x/n = 2)] & (2) \\
&\longrightarrow_{\delta} [(\lambda q. \text{if } \text{false true } q) \text{true}] & (3) \\
&\longrightarrow_{\beta} [\text{if } \text{false true true}] & (4) \\
&\longrightarrow_{\delta} \text{true} & (5)
\end{aligned}$$

Step (1) is valid because \square is a reduction context (by DELTA) and so $(\lambda p. \lambda q. \text{if } p \text{ true } q) \square$ is a reduction context (by GO RIGHT) and so $(\lambda p. \lambda q. \text{if } p \text{ true } q) \square (x/n = 2)$ is a reduction context (by GO LEFT).

Step (2) is valid because \square is a reduction context (by BETA) and so $\square (x/n = 2)$ is a reduction context (by GO LEFT).

Step (3) is valid because \square is a reduction context (by DELTA) and so $(\lambda q. \text{if } \text{false true } q) \square$ is a reduction context (by GO RIGHT).

Step (4) is valid because \square is a reduction context (by BETA).

Step (5) is valid because \square is a reduction context (by DELTA).

b. Redo the evaluation of the above expression for the values $x = 4$ and $n = 0$.

$$\begin{aligned}
\text{or } (n = 0) (x/n = 2) &= (\lambda p. \lambda q. \text{if } p \text{ true } q) [(n = 0)] (x/n = 2) \\
&\longrightarrow_{\delta} [(\lambda p. \lambda q. \text{if } p \text{ true } q) \text{true}] (x/n = 2) & (1) \\
&\longrightarrow_{\beta} (\lambda q. \text{if } \text{true true } q) [(x/n = 2)] & (2) \\
&\longrightarrow_{\delta} \text{error - division by zero} & (3)
\end{aligned}$$

Exercise 3: Reflection on Call-by-value

Consider the fact that all primitive functions are called by “value”, i.e., their arguments are evaluated ahead of time. In particular, the primitive function “if” acts in this way. Do you foresee any problems with this scheme? Write down a recursive function definition, such as that of factorial, and contemplate how its evaluation would proceed in this regime.

Having “if” evaluate its argument ahead of time means that it cannot be used to terminate a chain of recursive calls. Consider the following recursive definition:

$$\text{factorial } n = \text{if } (n = 0) 1 (n * (\text{factorial } (n - 1)))$$

If the evaluation of “if” requires the evaluation of both the “then” part and the “else” part then the recursion will never terminate, because the “else” part will be evaluated even when n reaches zero and the value of the recursive call is not actually required.

CBV languages must treat “if” as a special operator to avoid this problem.

Exercise 4: Delayed evaluation

Scheme is a call-by-value programming language. However, it introduced two constructs called **delay** and **force** in order to facilitate the use of infinite data structures. The syntax is:

$$M ::= x \mid c \mid \lambda x. M' \mid M_1 M_2 \mid \text{delay } M' \mid \text{force } M'$$

The idea is that the evaluation of (**delay** M') does nothing. (It represents the “delaying” of the evaluation of M' .) When the **force** operation is applied to a **delay**-term, then the evaluation of the delayed term is carried out. The reduction rule for this is:

$$\text{force } (\text{delay } M) \longrightarrow M$$

Using these constructs, the function for an infinite list of integers can be defined as follows:

$$\text{ints } n = \text{cons } n (\text{delay } (\text{ints } (n + 1)))$$

a. Propose a definition for the call-by-value reduction contexts for this extended language.

$$\mathcal{R} ::= \square \mid (\mathcal{R} M) \mid (V \mathcal{R}) \mid (c V_1 \dots \mathcal{R} \dots M_n) \mid \text{force } \mathcal{R}$$

Call the big-step evaluation rule which says that to evaluate **force** M involves evaluating M the FORCE rule. We also need to extend the syntax of values to include **delay** terms:

$$V ::= \lambda x. M \mid c \mid (c V_1 \dots V_k) \mid (\text{cons } V_1 V_2) \mid \mathbf{delay } M$$

where k is strictly less than the arity of c .

- b. Verify that your design allows terms such as $\text{car } (\mathbf{force } (\text{cdr } (\text{ints } 2)))$ to be reduced to value terms (with evaluation terminating finitely).

We will treat the names of functions as *values*, since they will have been substituted by lambda abstractions by this point. We write $\rightarrow_{F/D}$ for the $\mathbf{force } (\mathbf{delay } M) \rightarrow M$ reduction rule. Call the big-step evaluation rule which invokes $\rightarrow_{F/D}$ (analogous to the BETA rule for functions and the DELTA rule for function constants) the F/D rule.

$$\begin{aligned} \text{car } (\mathbf{force } (\text{cdr } [(ints\ 2)])) &\rightarrow_{\beta} \text{car } (\mathbf{force } [(cdr (\text{cons } 2 (\mathbf{delay } (\text{ints } (2 + 1))))])) && (1) \\ &\rightarrow_{\delta} \text{car } [(\mathbf{force } (\mathbf{delay } (\text{ints } (2 + 1))))] && (2) \\ &\rightarrow_{F/D} \text{car } (\text{ints } [(2 + 1)]) && (3) \\ &\rightarrow_{\delta} \text{car } [(ints\ 3)] && (4) \\ &\rightarrow_{\beta} [\text{car } (\text{cons } 3 (\mathbf{delay } (\text{ints } (3 + 1))))] && (5) \\ &\rightarrow_{\delta} 3 && (6) \end{aligned}$$

Step (1) is valid because $[]$ is a reduction context (by DELTA), and so $(\mathbf{force } [])$ is a reduction context (by FORCE), and so $\text{car } (\mathbf{force } [])$ is a reduction context (by GO RIGHT).

Step (2) is valid because $[]$ is a reduction context (by F/D), and so $(\text{car } [])$ is a reduction context (by GO RIGHT).

Step (3) is valid because $[]$ is a reduction context (by DELTA), and so $(\text{ints } [])$ is a reduction context (by GO RIGHT), and so $(\text{car } (\text{ints } []))$ is a reduction context (by GO RIGHT again).

Step (4) is valid because $[]$ is a reduction context, and so $(\text{car } [])$ is a reduction context (by GO RIGHT).

Step (5) is valid because $[]$ is a reduction context (by DELTA).

- c. Suppose we regard \mathbf{delay} and \mathbf{force} as syntactic sugar for basic lambda calculus terms as follows:

$$\begin{aligned} \mathbf{delay } M &= \lambda x. M \\ \mathbf{force } N &= N\ 0 \end{aligned}$$

Using the standard set-up for call-by-value, do we get the desired results for \mathbf{delay} and \mathbf{force} ?

Yes. This is easily verified for the above example:

$$\begin{aligned} \text{car } ((\text{cdr } [(ints\ 2)])\ 0) &\rightarrow_{\beta} \text{car } (((\text{cdr } (\text{cons } 2 (\lambda x. \text{ints } (2 + 1))))\ 0) && \text{GO RIGHT; GO LEFT; GO RIGHT; BETA} \\ &\rightarrow_{\delta} \text{car } [(\lambda x. \text{ints } (2 + 1))\ 0] && \text{GO RIGHT; GO LEFT; DELTA} \\ &\rightarrow_{\beta} \text{car } (\text{ints } [(2 + 1)]) && \text{GO RIGHT; BETA} \\ &\rightarrow_{\delta} \text{car } [(ints\ 3)] && \text{GO RIGHT; GO RIGHT; DELTA} \\ &\rightarrow_{\beta} [\text{car } (\text{cons } 3 (\lambda x. (\text{ints } (3 + 1))))] && \text{GO RIGHT; BETA} \\ &\rightarrow_{\delta} 3 && \text{DELTA} \end{aligned}$$