

C.S. 430 Assignment 6, Sample Solutions

Collin Jackson and Antony Courtney

Nov. 17, 2003

Note that these are *sample* solutions only; in many cases there were many acceptable answers.

Sample solutions provided courtesy of Collin Jackson.

1 Reynolds Problem 10.1

(20 points)

1.1 Normal-order Reductions

1.1.1

$$\begin{aligned} & (\lambda f. \lambda x. f(fx)) (\lambda b. \lambda x. \lambda y. b y x) (\lambda z. \lambda w. z) \Rightarrow \\ & (\lambda x. (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') x)) (\lambda z. \lambda w. z) \Rightarrow \\ & (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') (\lambda z. \lambda w. z)) \Rightarrow \\ & \lambda x'. \lambda y. ((\lambda b. \lambda x''. \lambda y'. b y' x'') (\lambda z. \lambda w. z)) y x' \Rightarrow \quad (\text{canonical form}) \\ & \lambda x'. \lambda y. (\lambda x''. \lambda y'. (\lambda z. \lambda w. z) y' x'') y x' \Rightarrow \\ & \lambda x'. \lambda y. (\lambda y'. (\lambda z. \lambda w. z) y' y) x' \Rightarrow \\ & \lambda x'. \lambda y. (\lambda z. \lambda w. z) x' y \Rightarrow \\ & \lambda x'. \lambda y. (\lambda w. x') y \Rightarrow \\ & \lambda x'. \lambda y. x' = \\ & \text{TRUE} \end{aligned}$$

1.1.2

$$\begin{aligned} & (\lambda d. d d) (\lambda f. \lambda x. f(fx)) \Rightarrow \\ & (\lambda f. \lambda x. f(fx)) (\lambda f. \lambda x. f(fx)) \Rightarrow \\ & \lambda x. (\lambda f. \lambda x'. f(fx')) ((\lambda f. \lambda x'. f(fx')) x) \Rightarrow \quad (\text{canonical form}) \\ & \lambda x. \lambda x'. ((\lambda f. \lambda x''. f(fx'')) x) (((\lambda f. \lambda x''. f(fx'')) x) x') \Rightarrow \\ & \lambda x. \lambda x'. (\lambda x''. x(x x'')) (((\lambda f. \lambda x''. f(fx'')) x) x') \Rightarrow \\ & \lambda x. \lambda x'. x(x(((\lambda f. \lambda x''. f(fx'')) x) x')) \Rightarrow \\ & \lambda x. \lambda x'. x(x((\lambda x''. x(x x'')) x')) \Rightarrow \end{aligned}$$

$$\lambda x. \lambda x'. x (x (x x')) =$$

*NUM*₄

1.2 Eager Reductions

1.2.1

$$\begin{aligned} & (\lambda f. \lambda x. f(fx)) (\lambda b. \lambda x. \lambda y. byx) (\lambda z. \lambda w. z) \Rightarrow \\ & (\lambda x. (\lambda b. \lambda x'. \lambda y. byx') ((\lambda b. \lambda x'. \lambda y. byx') x)) (\lambda z. \lambda w. z) \Rightarrow \\ & (\lambda b. \lambda x. \lambda y. byx) ((\lambda b. \lambda x. \lambda y. byx) (\lambda z. \lambda w. z)) \Rightarrow \\ & (\lambda b. \lambda x. \lambda y. byx) (\lambda x. \lambda y. (\lambda z. \lambda w. z) yx) \Rightarrow \\ & \lambda x. \lambda y. (\lambda x'. \lambda y'. (\lambda z. \lambda w. z) y' x') yx \Rightarrow \quad \text{(canonical form)} \\ & \lambda x. \lambda y. (\lambda y'. (\lambda z. \lambda w. z) y' y) x \Rightarrow \\ & \lambda x. \lambda y. (\lambda z. \lambda w. z) xy \Rightarrow \\ & \lambda x. \lambda y. (\lambda w. x) y \Rightarrow \\ & \lambda x. \lambda y. x = \\ & \text{TRUE} \end{aligned}$$

1.2.2

$$\begin{aligned} & (\lambda d. d d) (\lambda f. \lambda x. f(fx)) \Rightarrow \\ & (\lambda f. \lambda x. f(fx)) (\lambda f. \lambda x. f(fx)) \Rightarrow \\ & \lambda x. (\lambda f. \lambda x'. f(fx')) ((\lambda f. \lambda x'. f(fx')) x) \Rightarrow \quad \text{(canonical form)} \\ & \lambda x. (\lambda f. \lambda x'. f(fx')) (\lambda x'. x(xx')) \Rightarrow \\ & \lambda x. \lambda x'. (\lambda x''. x(xx'')) ((\lambda x''. x(xx'')) x') \Rightarrow \\ & \lambda x. \lambda x'. (\lambda x''. x(xx'')) (x(xx')) \Rightarrow \\ & \lambda x. \lambda x'. x(x(x(xx'))) = \\ & \text{NUM}_4 \end{aligned}$$

1.3 Proofs of Normal-order Evaluations

We use the indented proof style.

1.3.1

$$\begin{aligned} & (\lambda f. \lambda x. f(fx)) (\lambda b. \lambda x. \lambda y. byx) (\lambda z. \lambda w. z) \\ & \quad (\lambda f. \lambda x. f(fx)) (\lambda b. \lambda x. \lambda y. byx) \\ & \quad \quad \lambda f. \lambda x. f(fx) \Rightarrow \lambda f. \lambda x. f(fx) \\ & \quad \quad \quad (\lambda x. (\lambda b. \lambda x'. \lambda y. byx') ((\lambda b. \lambda x'. \lambda y. byx') x)) \Rightarrow (\lambda x. (\lambda b. \lambda x'. \lambda y. byx') ((\lambda b. \lambda x'. \lambda y. byx') x)) \\ & \Rightarrow (\lambda x. (\lambda b. \lambda x'. \lambda y. byx') ((\lambda b. \lambda x'. \lambda y. byx') x)) \\ & \quad (\lambda b. \lambda x'. \lambda y. byx') ((\lambda b. \lambda x'. \lambda y. byx') (\lambda z. \lambda w. z)) \\ & \quad \quad \lambda b. \lambda x'. \lambda y. byx' \Rightarrow \lambda b. \lambda x'. \lambda y. byx' \\ & \quad \quad \quad \lambda x'. \lambda y. ((\lambda b. \lambda x''. \lambda y'. by' x'') (\lambda z. \lambda w. z)) yx' \Rightarrow \lambda x'. \lambda y. ((\lambda b. \lambda x''. \lambda y'. by' x'') (\lambda z. \lambda w. z)) yx' \\ & \Rightarrow \lambda x'. \lambda y. ((\lambda b. \lambda x''. \lambda y'. by' x'') (\lambda z. \lambda w. z)) yx' \\ & \Rightarrow \lambda x'. \lambda y. ((\lambda b. \lambda x''. \lambda y'. by' x'') (\lambda z. \lambda w. z)) yx' \end{aligned}$$

1.3.2

$$\begin{aligned} & (\lambda d. d d) (\lambda f. \lambda x. f (f x)) \\ & \lambda d. d d \Rightarrow \lambda d. d d \\ & (\lambda f. \lambda x. f (f x)) (\lambda f. \lambda x. f (f x)) \\ & \lambda f. \lambda x. f (f x) \Rightarrow \lambda f. \lambda x. f (f x) \\ & \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \\ & \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \\ & \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \end{aligned}$$

1.4 Proofs of Eager Evaluations

We use the indented proof style.

1.4.1

$$\begin{aligned} & (\lambda f. \lambda x. f (f x)) (\lambda b. \lambda x. \lambda y. b y x) (\lambda z. \lambda w. z) \\ & (\lambda f. \lambda x. f (f x)) (\lambda b. \lambda x. \lambda y. b y x) \\ & \lambda f. \lambda x. f (f x) \Rightarrow \lambda f. \lambda x. f (f x) \\ & \lambda b. \lambda x. \lambda y. b y x \Rightarrow \lambda b. \lambda x. \lambda y. b y x \\ & (\lambda x. (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') x)) \Rightarrow (\lambda x. (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') x)) \\ & \Rightarrow (\lambda x. (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') x)) \\ & \lambda z. \lambda w. z \Rightarrow \lambda z. \lambda w. z \\ & (\lambda b. \lambda x'. \lambda y. b y x') ((\lambda b. \lambda x'. \lambda y. b y x') (\lambda z. \lambda w. z)) \\ & \lambda b. \lambda x'. \lambda y. b y x' \Rightarrow \lambda b. \lambda x'. \lambda y. b y x' \\ & (\lambda b. \lambda x'. \lambda y. b y x') (\lambda z. \lambda w. z) \\ & \lambda b. \lambda x'. \lambda y. b y x' \Rightarrow \lambda b. \lambda x'. \lambda y. b y x' \\ & \lambda z. \lambda w. z \Rightarrow \lambda z. \lambda w. z \\ & \lambda x'. \lambda y. (\lambda z. \lambda w. z) y x' \Rightarrow \lambda x'. \lambda y. (\lambda z. \lambda w. z) y x' \\ & \Rightarrow \lambda x'. \lambda y. (\lambda z. \lambda w. z) y x' \\ & \lambda x. \lambda y. (\lambda x'. \lambda y'. (\lambda z. \lambda w. z) y' x') y x \Rightarrow \lambda x. \lambda y. (\lambda x'. \lambda y'. (\lambda z. \lambda w. z) y' x') y x \\ & \Rightarrow \lambda x. \lambda y. (\lambda x'. \lambda y'. (\lambda z. \lambda w. z) y' x') y x \\ & \Rightarrow \lambda x. \lambda y. (\lambda x'. \lambda y'. (\lambda z. \lambda w. z) y' x') y x \end{aligned}$$

1.4.2

$$\begin{aligned} & (\lambda d. d d) (\lambda f. \lambda x. f (f x)) \\ & \lambda d. d d \Rightarrow \lambda d. d d \\ & (\lambda f. \lambda x. f (f x)) \Rightarrow (\lambda f. \lambda x. f (f x)) \\ & (\lambda f. \lambda x. f (f x)) (\lambda f. \lambda x. f (f x)) \\ & \lambda f. \lambda x. f (f x) \Rightarrow \lambda f. \lambda x. f (f x) \\ & \lambda f. \lambda x. f (f x) \Rightarrow \lambda f. \lambda x. f (f x) \\ & \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \\ & \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \\ & \Rightarrow \lambda x. (\lambda f. \lambda x'. f (f x')) ((\lambda f. \lambda x'. f (f x')) x) \end{aligned}$$

2 Reynolds Problem 10.14

(20 points)

We begin with the following definitions:

$$\begin{aligned}
 EXP &\stackrel{\text{def}}{=} \lambda m. \lambda n. n \ m \\
 NUM_n &\stackrel{\text{def}}{=} \lambda f. \lambda x. f^n \ x \\
 NUM_1 &\stackrel{\text{def}}{=} \lambda f. f \\
 MULT &\stackrel{\text{def}}{=} \lambda m. \lambda n. \lambda f. m \ (n \ f) \\
 THINGY_{m,n} &\stackrel{\text{def}}{=} \lambda f. \lambda x. (NUM_m^n \ f)^m \ x
 \end{aligned}$$

Our proof of $EXP \ NUM_m \ NUM_n \rightarrow^* \ NUM_{(m^n)}$ is by induction on n .

BASIS CASE

We apply transitive closure to the following steps.

$$\begin{aligned}
 EXP \ NUM_m \ NUM_0 &= \\
 (\lambda m. \lambda n. n \ m) (\lambda f. \lambda x. f^m \ x) (\lambda f. \lambda x. x) &\rightarrow \\
 (\lambda n. n \ (\lambda f. \lambda x. f^m \ x)) (\lambda f. \lambda x. x) &\rightarrow \\
 (\lambda f. \lambda x. x) (\lambda f. \lambda x. f^m \ x) &\rightarrow \\
 \lambda x. x \equiv \lambda f. f \stackrel{\text{def}}{=} NUM_1 &
 \end{aligned}$$

INDUCTIVE STEP

We assume our inductive hypothesis

$$EXP \ NUM_m \ NUM_n \rightarrow^* \ NUM_{(m^n)}$$

and make the following remarks:

Remark 2.1. $MULT \ NUM_m \ NUM_n \rightarrow^* \ NUM_{m \times n}$.

Proof: $MULT$ is presented on Reynolds p. 218 as an isomorphic equivalent for \times .

Remark 2.2. $MULT \ NUM_m \ (EXP \ NUM_m \ NUM_n) \rightarrow^* \ MULT \ NUM_m \ NUM_{m^n}$

Proof: By applying contextual closure for every step in the normal-order reduction provided by the inductive hypothesis, and then applying transitive closure to the derived steps.

Remark 2.3. $MULT \ NUM_m \ (EXP \ NUM_m \ NUM_n) \rightarrow^* \ NUM_{m^{n+1}}$

Proof: By Remarks 2.1 and 2.2, and by the definition of mathematical exponentiation on Reynolds p. 221.

Remark 2.4. $MULT \ NUM_m \ (EXP \ NUM_m \ NUM_n) \Rightarrow THINGY_{m,n}$

Proof: We give a normal-order reduction. The proof of this reduction is omitted.

$$\begin{aligned}
& MULT\ NUM_m (EXP\ NUM_m\ NUM_n) = \\
& (\lambda m. \lambda n. \lambda f. m (n f))\ NUM_m (EXP\ NUM_m\ NUM_n) \Rightarrow \\
& (\lambda n. \lambda f. NUM_m (n f))\ (EXP\ NUM_m\ NUM_n) \Rightarrow \\
& \lambda f. NUM_m (EXP\ NUM_m\ NUM_n f) = \\
& \lambda f. (\lambda f'. \lambda x. f^m x)\ (EXP\ NUM_m\ NUM_n f) \Rightarrow \\
& \lambda f. \lambda x. (EXP\ NUM_m\ NUM_n f)^m x = \\
& \lambda f. \lambda x. ((\lambda m. \lambda n. n m)\ NUM_m\ NUM_n f)^m x \Rightarrow \\
& \lambda f. \lambda x. (\lambda n. n\ NUM_m)\ NUM_n f)^m x \Rightarrow \\
& \lambda f. \lambda x. (NUM_n\ NUM_m f)^m x = \\
& \lambda f. \lambda x. ((\lambda f'. \lambda x'. f^n x')\ NUM_m f)^m x \Rightarrow \\
& \lambda f. \lambda x. ((\lambda x'. NUM_m^n x')\ f)^m x \Rightarrow \\
& \lambda f. \lambda x. (NUM_m^n f)^m x = \\
& THINGY_{m,n}
\end{aligned}$$

Remark 2.5. $EXP\ NUM_m\ NUM_{n+1} \rightarrow^* THINGY_{m,n}$

Proof: We apply transitive closure to the following steps.

$$\begin{aligned}
& EXP\ NUM_m\ NUM_{n+1} = \\
& (\lambda m. \lambda n. n m)\ NUM_m\ NUM_{n+1} \rightarrow \\
& (\lambda n. n\ NUM_m)\ NUM_{n+1} \rightarrow \\
& NUM_{n+1}\ NUM_m = \\
& (\lambda f. \lambda x. f^{n+1} x)\ NUM_m \rightarrow \\
& \lambda x. NUM_m^{n+1} x = \\
& \lambda x. (\lambda f. \lambda x'. f^m x')\ NUM_m^n x \rightarrow \\
& \lambda x. \lambda x'. (NUM_m^n x)^m x' \equiv \\
& \lambda f. \lambda x. (NUM_m^n f)^m x = \\
& THINGY_{m,n}
\end{aligned}$$

Remark 2.6. $THINGY_{m,n} \rightarrow^* NUM_{m^{n+1}}$

Proof: A subsequence of a normal order reduction of

$$MULT\ NUM_m (EXP\ NUM_m\ NUM_n)$$

is provided in the proof of Remark 2.4. By Remark 2.3, the full reduction sequence eventually leads to the normal form $NUM_{m^{n+1}}$. We apply transitive closure to the remaining steps of this reduction sequence.

Remark 2.7. $EXP\ NUM_m\ NUM_{n+1} \rightarrow^* NUM_{(m^{n+1})}$

Proof: We apply transitive closure to Remarks 2.5 and 2.6. This remark concludes our inductive proof.

3 Coincidence Theorem

(15 points)

We prove that

If $\eta w = \eta' w$ for all $w \in \text{FV}(e)$, then $[[e]]\eta = [[e]]\eta'$.

by structural induction on e , with a case analysis on the constructors of the abstract grammar. Recall that the result e/δ of simultaneously substituting δv for each occurrence of each variable v in e is defined by

$$\begin{aligned} v/\delta &= \delta v \\ (e_0 e_1)/\delta &= (e_0/\delta)(e_1/\delta) \\ (\lambda v. e)/\delta &= \lambda v_{\text{new}}. (e/[\delta | v : v_{\text{new}}]), \end{aligned}$$

where

$$v_{\text{new}} \notin \bigcup_{w \in \text{FV}(e) - \{v\}} \text{FV}(\delta w).$$

3.1 Variables

If $e = v$, then $\text{FV}(e) = \{v\}$, and if $\eta v = \eta' v$, then by the semantic equation,

$$[[e]]\eta = \eta v = \eta' v = [[e]]\eta'.$$

3.2 Applications

If $e = e_0 e_1$, then $\text{FV}(e) = \text{FV}(e_0) \cup \text{FV}(e_1)$. If $\eta w = \eta' w$ for all $w \in \text{FV}(e)$, then $\eta w = \eta' w$ for all $w \in \text{FV}(e_0)$ and for all $w \in \text{FV}(e_1)$, so by our inductive hypothesis $[[e_0]]\eta = [[e_0]]\eta'$ and $[[e_1]]\eta = [[e_1]]\eta'$. It follows from the semantic equation that

$$\begin{aligned} [[e_0 e_1]]\eta &= \\ \phi([[e_0]]\eta)([[e_1]]\eta) &= \\ \phi([[e_0]]\eta')([[e_1]]\eta') &= \\ [[e_0 e_1]]\eta'. \end{aligned}$$

3.3 Abstractions

If $e = \lambda v. q$, then $\text{FV}(e) = \text{FV}(q) - \{v\}$. If $\eta w = \eta' w$ for all $w \in \text{FV}(e)$, then $\eta w = \eta' w$ for all $w \in \text{FV}(q)$ with the possible exception of v . Even in the case $\eta v \neq \eta' v$, $[[\lambda v. q]]\eta = \psi(\lambda x \in D_\infty. [[q]][\eta | v : x]) = \psi(\lambda x \in D_\infty. [[q]][\eta' | v : x]) = [[\lambda v. q]]\eta'$ with the use of the inductive hypothesis.

4 Substitution Theorem

(15 points)

We prove that

$$\text{If } [[\delta w]]\eta' = \eta w \text{ for all } w \in \text{FV}(e), \text{ then } [[e/\delta]]\eta' = [[e]]\eta$$

by structural induction on e , with a case analysis on the constructors of the abstract grammar.

4.1 Variables

Suppose $e = v$ for some variable v . Then $\text{FV}(e) = \{v\}$, so if $[[\delta w]]\eta' = \eta w$ for all $w \in \text{FV}(e)$, then $[[e/\delta]]\eta' = [[e]]\eta$.

4.2 Applications

If $e = e_0 e_1$, then $\text{FV}(e) = \text{FV}(e_0) \cup \text{FV}(e_1)$. If $[[\delta w]]\eta' = \eta w$ for all $w \in \text{FV}(e)$, then $[[\delta w]]\eta' w = \eta w$ for all $w \in \text{FV}(e_0)$ and for all $w \in \text{FV}(e_1)$, so by our inductive hypothesis $[[e_0/\delta]]\eta' = [[e_0]]\eta$ and $[[e_1/\delta]]\eta' = [[e_1]]\eta$. It follows from the semantic equation that

$$\begin{aligned} [[e_0 e_1/\delta]]\eta' &= \\ [[(e_0/\delta)(e_1/\delta)]]\eta' &= \\ \phi([[e_0/\delta]]\eta')([[e_1/\delta]]\eta') &= \\ \phi([[e_0]]\eta)([[e_1]]\eta) &= \\ [[e_0 e_1]]\eta. & \end{aligned}$$

4.3 Abstractions

Suppose e is $\lambda v. q$, and $\eta q = [[\delta w]]\eta'$ holds for all $w \in \text{FV}(e)$. Then the definition of substitution and the semantic equation for abstraction give

$$\begin{aligned} [[(\lambda v. q)/\delta]]\eta' &= \\ [[\lambda v_{\text{new}}. (q/[\delta | v : v_{\text{new}}])]]\eta' &= \\ \psi(\lambda x \in D_{\infty}. [[(q/[\delta | v : v_{\text{new}}])]][\eta | v_{\text{new}} : x]) & \quad (*) \end{aligned}$$

Now consider the equation

$$[\eta | v : x]w = [[[\delta | v : v_{\text{new}}]w]][\eta' | v_{\text{new}} : x]$$

This equation holds when $w = v$, and it also holds for all $w \in \text{FV}(q) - \{v\}$, since then $v_{\text{new}} \notin \text{FV}(\delta w)$, and therefore $\eta w = [[\delta w]]\eta' = [[\delta w]][\eta' | v_{\text{new}} : x]$. Thus the equation holds for all $w \in \text{FV}(q)$ and, by the induction hypothesis, line (*) above equals

$$\psi(\lambda x \in D_{\infty}. [[q]][\eta | v : n]) = [[\lambda v. q]]\eta$$

5 Lists in the lambda calculus

(15 points)

We define

$$CONS \stackrel{\text{def}}{=} \lambda x. \lambda y. \lambda z. z \times y$$

$$CAR \stackrel{\text{def}}{=} \lambda x. x \lambda y. \lambda z. y$$

$$CDR \stackrel{\text{def}}{=} \lambda x. x \lambda y. \lambda z. z.$$

We can translate these definitions into Haskell as follows:

```
> cons = Lam "x" (Lam "y" (Lam "z" (App (App (Var "z") (Var "x")) (Var "y"))))
> car = Lam "x" (App (Var "x") (Lam "y" (Lam "z" (Var "y"))))
> cdr = Lam "x" (App (Var "x") (Lam "y" (Lam "z" (Var "z"))))
```

Then our tests cases are:

```
> pair = App (App cons (Var "blue pill")) (Var "red pill")
> head = App car pair      -- returns "blue pill"
> tail = App cdr pair      -- returns "red pill"
```

6 Infinite lists

(15 points)

Some students chose to interpret this question by using the eager version of the Y combinator, Y_v for tracing eager evaluation, which produces the correct result instead of diverging. This was perfectly acceptable. The following sample solution, however, traces the normal-order Y combinator under both normal-order and eager evaluation:

$$\begin{aligned} ONES &\stackrel{\text{def}}{=} Y (CONS ONE) = \\ &(\lambda f. (\lambda x. f(xx))(\lambda x. f(xx)))(CONS ONE) \rightarrow \\ &(\lambda x. (CONS ONE)(xx))(\lambda x. (CONS ONE)(xx)) \rightarrow \\ &CONS ONE((\lambda x. (CONS ONE)(xx))(\lambda x. (CONS ONE)(xx))) \end{aligned}$$

We note that $CONS ONE ONES$ also reduces to this expression:

$$\begin{aligned} CONS ONE ONES &= \\ CONS ONE Y (CONS ONE) &= \\ CONS ONE (\lambda f. (\lambda x. f(xx))(\lambda x. f(xx))) (CONS ONE) &\rightarrow \\ CONS ONE ((\lambda x. (CONS ONE)(xx))(\lambda x. (CONS ONE)(xx))) &. \end{aligned}$$

Using normal-order evaluation,

$$\begin{aligned}
& \text{CAR ONES} = \\
& \text{CAR } (Y \text{ (CONS ONE)}) = \\
& (\lambda x. x \lambda y. \lambda z. y) (Y \text{ (CONS ONE)}) \rightarrow \\
& (Y \text{ (CONS ONE)}) \lambda y. \lambda z. y = \\
& ((\lambda f. (\lambda x. f(\text{xx})))(\lambda x. f(\text{xx})))(\text{CONS ONE}) \lambda y. \lambda z. y \rightarrow \\
& (\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx})) \lambda y. \lambda z. y \rightarrow \\
& (\text{CONS ONE}(\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx}))) \lambda y. \lambda z. y = \\
& (\lambda x. \lambda y. \lambda z. z \times y) \text{ ONE } ((\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx}))) \lambda y. \lambda z. y \rightarrow \\
& (\lambda y. \lambda z. z \text{ ONE } y)(\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx})) \lambda y. \lambda z. y \rightarrow \\
& \lambda z. z \text{ ONE } (\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx})) \lambda y. \lambda z. y \\
& (\lambda y. \lambda z. y) \text{ ONE } (\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx})) \rightarrow \\
& (\lambda z. \text{ONE}) (\lambda x. (\text{CONS ONE})(\text{xx}))(\lambda x. (\text{CONS ONE})(\text{xx})) \\
& \text{ONE},
\end{aligned}$$

which is the result we'd expect in a language like Haskell. However, eager evaluation is less successful:

$$\begin{aligned}
& \text{CAR ONES} = \\
& \text{CAR } (Y \text{ (CONS ONE)}) = \\
& \text{CAR } (Y ((\lambda x. \lambda y. \lambda z. z \times y) \text{ ONE})) \rightarrow \\
& \text{CAR } (Y (\lambda y. \lambda z. z \text{ ONE } y)) = \\
& \text{CAR } ((\lambda f. (\lambda x. f(\text{xx})))(\lambda x. f(\text{xx}))) (\lambda y. \lambda z. z \text{ ONE } y) \rightarrow \\
& \text{CAR } ((\lambda x. ((\lambda y. \lambda z. z \text{ ONE } y)(\text{xx}))(\lambda x. (\lambda y. \lambda z. z \text{ ONE } y)(\text{xx}))) \rightarrow \\
& \text{CAR } ((\lambda y. \lambda z. z \text{ ONE } y)((\lambda x. (\lambda y. \lambda z. z \text{ ONE } y)(\text{xx})) (\lambda x. (\lambda y. \lambda z. z \text{ ONE } y)(\text{xx})))
\end{aligned}$$

The expression

$$(\lambda x. (\lambda y. \lambda z. z \text{ ONE } y)(\text{xx})) (\lambda x. (\lambda y. \lambda z. z \text{ ONE } y)(\text{xx}))$$

must be expanded to proceed, but the expression appears within its own evaluation and must be evaluated in that context as well. There is no end to the recursion, so the eager evaluation diverges.