A Solutions to Exercises

Solution to Exercise 2.4

We calculate the sets of free and bound variables:

$$
\begin{array}{ll}
 & FV((\lambda y.(y\ x))\ (\lambda x.(x\ y))\ (\lambda z.(z\ x\ y))) \\
= & FV((\lambda y.(y\ x))\ (\lambda x.(x\ y)))\cup FV(\lambda z.(z\ x\ y)) \\
= & FV(\lambda y.(y\ x))\cup FV(\lambda x.(x\ y))\cup FV(\lambda z.(z\ x\ y)) \\
= & (FV((y\ x))\setminus\{y\})\cup(FV((x\ y))\setminus\{x\})\cup(FV((z\ x\ y))\setminus\{z\}) \\
= & (FV(y)\cup FV(x))\setminus\{y\})\cup(FV(x)\cup FV(y))\setminus\{x\})\cup(FV(z)\cup FV(x)\cup FV(y))\setminus\{z\}) \\
= & (\{x,y\})\setminus\{y\})\cup(\{x,y\}\setminus\{x\})\cup(\{x,y,z\}\setminus\{z\}) \\
= & \{x,y\}
\end{array}
$$

$$
\begin{array}{ll}
 & BV((\lambda y.(y\ x))\ (\lambda x.(x\ y))\ (\lambda z.(z\ x\ y))) \\
= & BV((\lambda y.(y\ x))\ (\lambda x.(x\ y)))\cup BV(\lambda z.(z\ x\ y)) \\
= & BV(\lambda y.(y\ x))\cup BV(\lambda x.(x\ y))\cup BV(\lambda z.(z\ x\ y)) \\
= & (BV((y\ x))\cup\{y\})\cup(BV((x\ y))\cup\{x\})\cup(BV((z\ x\ y))\cup\{z\}) \\
= & (BV(y)\cup BV(x))\cup\{y\})\cup(BV(x)\cup BV(y))\cup\{x\})\cup(BV(z)\cup BV(x)\cup BV(y))\setminus\{z\}) \\
= & (\emptyset\cup\{y\})\cup(\emptyset\cup\{x\})\cup(\emptyset\cup\{z\}) \\
= & \{x,y,z\}
\end{array}
$$

We underline the bound occurrences:

$$(\lambda y.(\underline{y}\ x))\ (\lambda x.(\underline{x}\ y))\ (\lambda z.(\underline{z}\ x\ y))$$

and the free occurrences:

$$(\lambda y.(y\ \underline{x}))\ (\lambda x.(x\ \underline{y}))\ (\lambda z.(z\ \underline{x}\ \underline{y}))$$

of variables $\{x,y,z\}$.

An α-equivalent expression which fulfills the distinct variable convention can be obtained by renaming each variable at a binder by a fresh name, e.g.

$$(\lambda x_1.(x_1\ x))\ (\lambda x_2.(x_2\ y))\ (\lambda x_3.(x_3\ x\ y))$$

Solution to Exercise 2.5

Let $e_1 = (\lambda w.w)\ (\lambda x.x)\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z))$. We write down all reduction contexts $R \in R_\lambda$ and expressions e_2 s.t. $R[e_2] = e_1$:

- $R = [\cdot]$ and $e_2 = (\lambda w.w)\ (\lambda x.x)\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z))$.
- $R = ([\cdot]\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z)))$ and $e_2 = (\lambda w.w)\ (\lambda x.x)$.
- $R = (([\cdot]\ (\lambda x.x))\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z)))$ and $e_2 = (\lambda w.w)$.

The normal order reduction step is a β-reduction applied to the expression inside the second context:

$$
\begin{array}{ll}
 & (\lambda w.w)\ (\lambda x.x)\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z)) \\
\xrightarrow{no,\beta} & w[(\lambda x.x)/w]\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z)) \\
= & (\lambda x.x)\ ((\lambda y.((\lambda u.y)\ y))\ (\lambda z.z))
\end{array}
$$

Solution to Exercise 2.7

We use the label shifting algorithm to find the normal order redex (only the final labeling is shown):

$$
\begin{array}{ll}
 & (\lambda y.(\lambda x.y\ (x\ x)))^{\star}\ (\lambda.y_1.(\lambda x_1.y_1\ (x_1\ x_1)))(\lambda w.\lambda z.w) \\
\xrightarrow{no,\beta} & (\lambda x.(\lambda.y_1.(\lambda x_1.y_1\ (x_1\ x_1)))\ (x\ x))^{\star}\ (\lambda w.\lambda z.w) \\
\xrightarrow{no,\beta} & (\lambda y_1.(\lambda x_1.y_1\ (x_1\ x_1)))^{\star}\ ((\lambda w.\lambda z.w)\ (\lambda w_1.\lambda z_1.w_1)) \\
\xrightarrow{no,\beta} & (\lambda x_1.((\lambda w.\lambda z.w)\ (\lambda w_1.\lambda z_1.w_1))\ (x_1\ x_1))^{\star}
\end{array}
$$

Solution to Exercise 2.17

To model pairs, we can define a type constructor *Pair* (of arity 2) with one data constructor `Pair` (of arity 2).

The list of three pairs of Boolean values $[(\texttt{True},\texttt{False}),(\texttt{True},\texttt{True}),(\texttt{False},\texttt{False})]$ is written as

`Cons (Pair True False) (Cons (Pair True True) (Cons (Pair False False) Nil))`.

Solution to Exercise 2.20

We use two nested `case`-expressions to compute the second element of a list. Note that we have to provide alternatives for the cases that the list is empty or contains only one element. In this case (which should be treated like errors) we return the diverging expression Ω, since it is usual to identify all errors with $\bot$.

$$
\begin{array}{l}
\lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{ \\
\quad \texttt{Nil} \to \Omega; \\
\quad (\texttt{Cons}\ y\ ys) \to \texttt{case}_{List}\ ys\ \texttt{of}\ \{ \\
\qquad\qquad \texttt{Nil} \to \Omega; \\
\qquad\qquad (\texttt{Cons}\ z\ zs) \to z\} \\
\}
\end{array}
$$

Solution to Exercise 2.29

$$
\begin{array}{ll}
 & ((\lambda xs.\mathsf{last}\ xs)^{\star}\ (\texttt{Cons}\ ((\lambda x.x)\ \texttt{True})\ \texttt{Nil})) \\
\xrightarrow{no,\beta} & (\mathsf{last}^{\star}\ (\texttt{Cons}\ ((\lambda x.x)\ \texttt{True})\ \texttt{Nil})) \\
\xrightarrow{no,SC\beta} & \texttt{case}_{List}\ (\texttt{Cons}\ ((\lambda x.x)\ \texttt{True})\ \texttt{Nil})^{\star}\ \texttt{of}\ \{ \\
 & \quad \texttt{Nil} \to \Omega; \\
 & \quad (\texttt{Cons}\ y\ ys) \to \texttt{case}_{List}\ ys\ \texttt{of}\ \{\texttt{Nil} \to y; (\texttt{Cons}\ u\ us) \to \mathsf{last}\ ys\}\} \\
\xrightarrow{no,case} & \texttt{case}_{List}\ \texttt{Nil}^{\star}\ \texttt{of}\ \{\texttt{Nil} \to ((\lambda x.x)\ \texttt{True}); (\texttt{Cons}\ u\ us) \to \mathsf{last}\ \texttt{Nil}\} \\
\xrightarrow{no,case} & ((\lambda x.x)^{\star}\ \texttt{True}) \\
\xrightarrow{no,\beta} & \texttt{True}
\end{array}
$$

Solution to Exercise 2.32

1. The in-equation $\texttt{True} \not\sim_c \texttt{False}$ holds, since for the context

$$C := \texttt{case}_{Bool}\ [\cdot]\ \{\texttt{True} \to \Omega; \texttt{False} \to \texttt{True}\},$$

the expression $C[\texttt{True}]$ diverges (i.e. $C[\texttt{True}]{\Uparrow}$) while the expression $C[\texttt{False}]$ converges (i.e. $C[\texttt{False}]{\Downarrow}$).

2. The in-equation $x \not\sim_c y$ holds, since for the context $C := (\lambda x, y.[\cdot])\ (\lambda z.z)\ \Omega$, it holds: $C[x]\Downarrow$ and $C[y]\Uparrow$.
3. The in-equation $\lambda x.\lambda y.y \not\sim_c \lambda x.\lambda y.\texttt{seq}\ x\ y$ holds, since for the context $C := ([\cdot]\ \Omega\ (\lambda z.z))$, we have $C[\lambda x.\lambda y.y]\Downarrow$ and $C[\lambda x.\lambda y.\texttt{seq}\ x\ y]\Uparrow$.
4. The in-equation $\mathsf{head} \not\sim_c \mathsf{tail}$ holds, since for the context, $C := ([\cdot]\ \texttt{Cons}\ \Omega\ \texttt{Nil})$ we have $C[\mathsf{head}] \xrightarrow{no,*} \Omega$ and thus $C[\mathsf{head}]\Uparrow$, while $C[\mathsf{tail}] \xrightarrow{no,*} \texttt{Nil}$ and thus $C[\mathsf{tail}]\Downarrow$.
5. The in-equation $\lambda xs.\texttt{Cons}\ (\mathsf{head}\ xs)\ (\mathsf{tail}\ \mathsf{xs}) \not\sim_c \lambda xs.xs$ holds, since for $C := ([\cdot]\ (\lambda z.z))$, we have $C[\lambda xs.\texttt{Cons}\ (\mathsf{head}\ xs)\ (\mathsf{tail}\ \mathsf{xs})]\Uparrow$ and $C[\lambda xs.xs]\Downarrow$.

Solution to Exercise 2.49

The transformation (seq-app) is defined as follows:

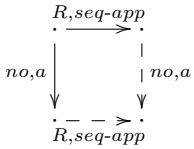
$$\text{(seq-app)} \quad ((\texttt{seq}\ e_1\ e_2)\ e_3) \to (\texttt{seq}\ e_1\ (e_2\ e_3))$$

We compute a complete set of forking diagrams for $(R, \textit{seq-app})$, i.e. we have to inspect all the overlappings of a (seq-app)-transformation applied in a reduction context with a normal order reduction. Thus it is sufficient to unify the left hand side of the normal order rules (i.e. the reduction rules applied in reduction contexts) against the left hand side of (seq-app) applied in a reduction context.

Let $R_1[(\texttt{seq}\ e_1\ e_2)\ e_3]$ be the left hand side of an $(R, \textit{seq-app})$-transformation.

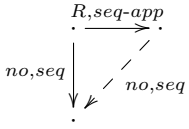
The general case is that $R_2[s] \xrightarrow{no} R_2[t]$ with redex $s \to t$. We split this case by the concrete reduction rule (instantiating s and t with more concrete expressions), if needed.

An easy case is, that $R_2 = R_1[(\texttt{seq}\ R_3\ e_2)\ e_3]$, since in this case the reduction and the transformation commute, i.e. the forking diagram

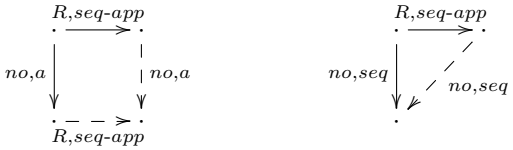


for $a \in \{(case), (seq), (SC\beta), (\beta)\}$

is applicable. For $a \neq (seq)$ this is the only possible case. For a (no,seq)-reduction there is the additional case that $R_2 = R_1[[\cdot]\ e_3]$. Then the following triangle diagram holds:

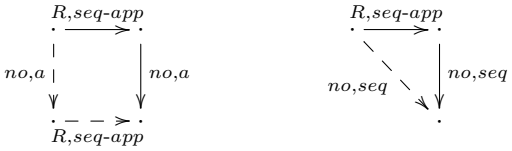


Thus, a complete set of forking diagrams for (R,seq-app) is:



for $a \in \{(case), (seq), (SC\beta), (\beta)\}$

A complete set of commuting diagrams for (R,seq-app) can be computed analogously. It is:



for $a \in \{(case), (seq), (SC\beta), (\beta)\}$

Also the following property holds, which can be used as an answer diagram: if $e_1 \xrightarrow{seq\text{-}app} e_2$, then $R[e_1]$ is a value iff $R[e_2]$ is a value. Thus, the answer diagram $A \xleftarrow{seq\text{-}app} \rightsquigarrow A$ and also the answer diagram $A \xrightarrow{seq\text{-}app} \rightsquigarrow A$ holds.

Now we show $R[\sigma(e_1)]{\Downarrow} \iff R[\sigma(e_2)]{\Downarrow}$ for any reduction context R and closing substitution σ and $e_1 \xrightarrow{seq-app} e_2$. Since the transformation is closed under substitutions the diagrams also hold for all substitutions. We now encode the diagrams as term rewrite system:

```
(VAR x)
(RULES
  T(no_beta(x))   -> no_beta(T(x))
  T(no_scbeta(x)) -> no_scbeta(T(x))
  T(no_case(x))   -> no_case(T(x))
  T(no_seq(x))    -> no_seq(T(x))
  T(no_seq(x))    -> no_seq(x)
  T(A)            -> A
)
```

where T represents $\xleftarrow{R,seq-app}$ for the forking case and $\xrightarrow{R,seq-app}$ for the commuting case. AProVE shows termination of the TRS.

Finally, the context lemma (Theorem 2.34) shows that (seq-app) is a correct program transformation.

Solution to Exercise 2.63

For proving correctness of the program transformation $\texttt{append Nil}\ xs \to xs$, it suffices to show $\texttt{append Nil}\ xs \sim_c xs$ for all expressions xs. Let xs be arbitrary but fixed, then

$$\begin{array}{ll} & \texttt{append Nil}\ xs \\ \xrightarrow{SC\beta} & \texttt{case}_{List}\ \texttt{Nil of}\ \{ \\ & \quad \texttt{Nil} \to xs; \\ & \quad (\texttt{Cons}\ u\ us) \to \texttt{Cons}\ u\ (\texttt{append}\ us\ xs)\} \\ \xrightarrow{case} & xs \end{array}$$

By Theorem 2.35 (SCβ) and (case) are correct program transformations and thus since $\sim_c$ is a congruence, we have $\texttt{append Nil}\ xs \sim_c xs$.

For proving correctness of the transformation $\texttt{append}\ xs\ \texttt{Nil} \to xs$, we define the following admissible predicate $\mathcal{P}(xs) := \texttt{append}\ xs\ \texttt{Nil} \sim_c xs$. We use the induction scheme from Theorem 2.59 for the predicate $\mathcal{P}$.

- **Case $\mathcal{P}(\bot)$:** We transform the left hand side of the equation into the right hand side of the equation:

$$\begin{array}{lll} & \texttt{append}\ \bot\ \texttt{Nil} & \\ \sim_c & \texttt{case}_{List}\ \bot\ \texttt{of}\ alts & \text{(since (SC}\beta\text{) is correct)} \\ \sim_c & \bot & \text{(since (case}\bot\text{) is correct)} \end{array}$$

- **Case $\mathcal{P}$(`Nil`):** We transform the left hand side of the equation into the right hand side of the equation:

$$\begin{array}{ll}
& \mathsf{append}\ \texttt{Nil}\ \texttt{Nil} \\
\sim_c & \texttt{case}_{List}\ \texttt{Nil}\ \texttt{of}\ alts & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \texttt{Nil} & \text{(since (case) is correct)}
\end{array}$$

- **Induction step:** As induction hypothesis we assume that for an arbitrary (but fixed) list expression xs, $\mathcal{P}(xs)$ holds, and show $\mathcal{P}(\texttt{Cons}\ x\ xs)$ where x is an arbitrary expression. We transform the left hand side of the equation into the right hand side of the equation:

$$\begin{array}{ll}
& \mathsf{append}\ (\texttt{Cons}\ x\ xs)\ \texttt{Nil} \\
\sim_c & \texttt{case}_{List}\ (\texttt{Cons}\ x\ xs)\ \texttt{of}\ \{ & \text{(since (SC}\beta\text{) is correct)} \\
& \quad \texttt{Nil} \to \texttt{Nil}; \\
& \quad (\texttt{Cons}\ u\ us) \to \texttt{Cons}\ u\ (\mathsf{append}\ us\ \texttt{Nil})\} \\
\sim_c & \texttt{Cons}\ x\ (\mathsf{append}\ xs\ \texttt{Nil}) & \text{(since (case) is correct)} \\
\sim_c & \texttt{Cons}\ x\ xs & \text{(induction hypothesis and since } \sim_c \text{ is a congruence)}
\end{array}$$

Solution to Exercise 2.67

Proving that the program transformation $\mathsf{reverse}\ (\mathsf{reverse}\ xs) \to xs$ is not correct for all lists xs requires a counter-example. I.e., a list xs, and a context C s.t. $C[xs]\Downarrow$ but $C[\mathsf{reverse}\ (\mathsf{reverse}\ xs)]\Uparrow$ (or vice versa). Intuitively, if xs is an infinite list, and C is the empty context, then xs converges, but $(\mathsf{reverse}\ (\mathsf{reverse}\ xs))\Downarrow$. However, it is easier to use a partial list (which also counts as a list expression). So, let $xs = \texttt{Cons True}\ \bot$. Then $xs\Downarrow$, but $\mathsf{reverse}\ (\mathsf{reverse}\ xs) \sim_c \bot$ and thus $\mathsf{reverse}\ (\mathsf{reverse}\ xs)\Uparrow$:

$$\begin{array}{ll}
& \mathsf{reverse}\ (\mathsf{reverse}\ (\texttt{Cons True}\ \bot)) \\
\sim_c & \mathsf{reverse}\ (\mathsf{append}\ (\mathsf{reverse}\ \bot)\ (\texttt{Cons True Nil})) \\
& & \text{(since (SC}\beta\text{) and (case) are correct)} \\
\sim_c & \mathsf{reverse}\ (\mathsf{append}\ (\texttt{case}_{List}\ \bot\ \texttt{of}\ alts)\ (\texttt{Cons True Nil})) & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \mathsf{reverse}\ (\mathsf{append}\ \bot\ (\texttt{Cons True Nil})) & \text{(since (case}\bot\text{) is correct)} \\
\sim_c & \mathsf{reverse}\ (\texttt{case}_{List}\ \bot\ \texttt{of}\ alts) & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \mathsf{reverse}\ \bot & \text{(since (case}\bot\text{) is correct)} \\
\sim_c & \texttt{case}_{List}\ \bot\ \texttt{of}\ alts & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \bot & \text{(since (case}\bot\text{) is correct)}
\end{array}$$

We prove that $\mathsf{reverse}\ (\mathsf{reverse}\ xs) \to xs$ is correct for all finite lists. We define the predicate $\mathcal{P}(xs) = \mathsf{reverse}\ (\mathsf{reverse}\ xs) \sim_c xs$ and apply the induction scheme for finite lists (Theorem 2.61):

- **Case $\mathcal{P}$(`Nil`):** We transform the left hand side of the equation into the right hand side of the equation:

$$\begin{array}{ll}
& \mathsf{reverse}\ (\mathsf{reverse}\ \texttt{Nil}) \\
\sim_c & \mathsf{reverse}\ (\texttt{case}_{List}\ \texttt{Nil}\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}, \ldots\}) & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \mathsf{reverse}\ \texttt{Nil} & \text{(since (case) is correct)} \\
\sim_c & \texttt{case}_{List}\ \texttt{Nil}\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}, \ldots\} & \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \texttt{Nil} & \text{(since (case) is correct)}
\end{array}$$

○ **Induction step:** As induction hypothesis we assume that for an arbitrary (but fixed) finite list expression xs, $\mathcal{P}(xs)$ holds, and show $\mathcal{P}(\texttt{Cons}\ x\ xs)$ where x is an arbitrary expression. We transform the left hand side of the equation into the right hand side of the equation:

$$
\begin{array}{ll}
& \mathsf{reverse}\ (\mathsf{reverse}\ (\texttt{Cons}\ x\ xs)) \\
\sim_c & \mathsf{reverse}(\mathsf{case}_{List}\ (\texttt{Cons}\ x\ xs)\ \texttt{of}\ \{ \qquad) \\
& \quad \texttt{Nil} \rightarrow \texttt{Nil}; \\
& \quad (\texttt{Cons}\ u\ us) \rightarrow (\mathsf{append}\ (\mathsf{reverse}\ us)\ (\texttt{Cons}\ u\ \texttt{Nil}))\} \\
& \hfill \text{(since (SC}\beta\text{) is correct)} \\
\sim_c & \mathsf{reverse}(\mathsf{append}\ (\mathsf{reverse}\ xs)\ (\texttt{Cons}\ x\ \texttt{Nil}))\} \quad \text{(since (case) is correct)} \\
\sim_c & \mathsf{append}\ (\mathsf{reverse}\ (\texttt{Cons}\ x\ \texttt{Nil}))\ (\mathsf{reverse}\ (\mathsf{reverse}\ xs)) \quad \text{(by Example 2.65)} \\
\sim_c & \mathsf{append}\ (\mathsf{reverse}\ (\texttt{Cons}\ x\ \texttt{Nil}))\ xs \quad \text{(by the induction hypothesis)} \\
\sim_c & \mathsf{append}\ (\texttt{Cons}\ x\ \texttt{Nil})\ xs \quad \text{(since (SC}\beta\text{) and (case) are correct)} \\
\sim_c & \texttt{Cons}\ x\ (\mathsf{append}\ \texttt{Nil}\ xs) \quad \text{(since (SC}\beta\text{) and (case) are correct)} \\
\sim_c & \texttt{Cons}\ x\ xs \quad \text{(since (SC}\beta\text{) and (case) are correct)}
\end{array}
$$

Solution to Exercise 3.10

We apply the labeling algorithm to the expression e:

$$
\begin{array}{ll}
& (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ head\ (rep\ \texttt{True}))^{\mathsf{top}} \\
\rightarrow & (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ (head\ (rep\ \texttt{True}))^{\mathsf{sub}})^{\mathsf{vis}} \\
\rightarrow & (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ (head^{\mathsf{sub}}\ (rep\ \texttt{True}))^{\mathsf{vis}})^{\mathsf{vis}} \\
\rightarrow & (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = (\lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\})^{\mathsf{sub}} \\
& \ \texttt{in}\ (head^{\mathsf{vis}}\ (rep\ \texttt{True}))^{\mathsf{vis}})^{\mathsf{vis}}
\end{array}
$$

The matching reduction is a (no,cp-in)-reduction and thus $e \xrightarrow{no,cp\text{-}in} e_1$ where

$$
\begin{array}{rl}
e_1 := & \texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ (\lambda xs'.\mathsf{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y'\ ys')\ \rightarrow\ y'\})\ (rep\ \texttt{True})
\end{array}
$$

We apply the labeling algorithm to e_1:

$$
\begin{array}{ll}
& (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ (\lambda xs'.\mathsf{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y'\ ys')\ \rightarrow\ y'\})\ (rep\ \texttt{True}))^{\mathsf{top}} \\
\rightarrow & (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ ((\lambda xs'.\mathsf{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y'\ ys')\ \rightarrow\ y'\})\ (rep\ \texttt{True}))^{\mathsf{sub}})^{\mathsf{vis}} \\
\rightarrow & (\texttt{letrec}\ rep = \lambda x.(\texttt{Cons}\ x\ rep), \\
& \qquad head = \lambda xs.\mathsf{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y\ ys)\ \rightarrow\ y\} \\
& \ \texttt{in}\ ((\lambda xs'.\mathsf{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \rightarrow \bot; (\texttt{Cons}\ y'\ ys')\ \rightarrow\ y'\})^{\mathsf{sub}}\ (rep\ \texttt{True}))^{\mathsf{vis}})^{\mathsf{vis}}
\end{array}
$$

The matching reduction is a (no,lbeta)-reduction and thus $e_1 \xrightarrow{no,lbeta} e_2$ where

$$\begin{array}{l} e_2 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\} \\ \qquad \texttt{in letrec } xs' = (rep\ \texttt{True})\ \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}) \end{array}$$

The labeling algorithm labels e_2 as follows:

$$\begin{array}{l} (\texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\} \\ \texttt{in}\ (\texttt{letrec } xs' = (rep\ \texttt{True})\ \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}))^{\mathsf{sub}})^{\mathsf{vis}} \end{array}$$

Thus, the normal order reduction is $e_2 \xrightarrow{no,llet\text{-}in} e_3$ where

$$\begin{array}{l} e_3 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (rep\ \texttt{True}) \\ \qquad \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}) \end{array}$$

The labeling algorithm labels e_3 as follows:

$$\begin{array}{l} (\texttt{letrec } rep = (\lambda x.(\texttt{Cons } x\ rep))^{\mathsf{sub}}, \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (rep^{\mathsf{vis}}\ \texttt{True})^{\mathsf{vis}} \\ \texttt{in}\ (\texttt{case}_{List}\ xs'^{\mathsf{vis}}\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\})^{\mathsf{vis}})^{\mathsf{vis}} \end{array}$$

The normal order reduction thus is $e_3 \xrightarrow{no,cp-e} e_4$ where

$$\begin{array}{l} e_4 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (\lambda x'.(\texttt{Cons } x'\ rep))\ \texttt{True}) \\ \qquad \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}) \end{array}$$

The labeling algorithm applied to e_4 results in

$$\begin{array}{l} (\texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (\lambda x'.(\texttt{Cons } x'\ rep))^{\mathsf{sub}}\ \texttt{True})^{\mathsf{vis}} \\ \texttt{in}\ (\texttt{case}_{List}\ xs'^{\mathsf{vis}}\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\})^{\mathsf{vis}})^{\mathsf{vis}} \end{array}$$

The normal order reduction is $e_4 \xrightarrow{no,lbeta} e_5$ where

$$\begin{array}{l} e_5 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = \texttt{letrec } x' = \texttt{True in Cons } x'\ rep \\ \qquad \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}) \end{array}$$

Applying the labeling algorithm to e_5 results in

$$\begin{array}{l} (\texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (\texttt{letrec } x' = \texttt{True in Cons } x'\ rep)^{\mathsf{sub}} \\ \texttt{in}\ (\texttt{case}_{List}\ xs'^{\mathsf{vis}}\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\})^{\mathsf{vis}})^{\mathsf{vis}} \end{array}$$

and thus $e_5 \xrightarrow{no,llet-e} e_6$ where

$$\begin{array}{l} e_6 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = \texttt{Cons}\ x'\ rep, \\ \qquad\qquad x' = \texttt{True} \\ \quad \texttt{in}\ (\texttt{case}_{List}\ xs'\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\}) \end{array}$$

The labeling of e_6 is:

$$\begin{array}{l} (\texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = (\texttt{Cons}\ x'\ rep)^{\mathsf{vis}}, \\ \qquad\qquad x' = \texttt{True} \\ \texttt{in}\ (\texttt{case}_{List}\ xs'^{\mathsf{vis}}\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y'\ ys')\ \to\ y'\})^{\mathsf{vis}})^{\mathsf{vis}} \end{array}$$

and thus $e_6 \xrightarrow{no,case-in} e_7$ where

$$\begin{array}{l} e_7 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = \texttt{Cons}\ u\ us, u = x', us = rep, \\ \qquad\qquad x' = \texttt{True} \\ \quad \texttt{in}\ (\texttt{letrec}\ y' = u, ys' = us\ \texttt{in}\ y') \end{array}$$

The labeling of e_7 is:

$$\begin{array}{l} (\texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = \texttt{Cons}\ u\ us, u = x', us = rep, \\ \qquad\qquad x' = \texttt{True} \\ \texttt{in}\ (\texttt{letrec}\ y' = u, ys' = us\ \texttt{in}\ y')^{\mathsf{sub}})^{\mathsf{vis}} \end{array}$$

and thus $e_7 \xrightarrow{no,llet-in} e_8$ where

$$\begin{array}{l} e_7 := \texttt{letrec } rep = \lambda x.(\texttt{Cons } x\ rep), \\ \qquad\qquad head = \lambda xs.\texttt{case}_{List}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \bot; (\texttt{Cons}\ y\ ys)\ \to\ y\}, \\ \qquad\qquad xs' = \texttt{Cons}\ u\ us, u = x', us = rep, \\ \qquad\qquad x' = \texttt{True}, y' = u, ys' = us \\ \quad \texttt{in}\ y' \end{array}$$

The expression e_8 is a WHNF, and thus $e{\Downarrow}$.

Solution to Exercise 3.26

The following term rewrite system:

```
(VAR x)
(RULES
 S_cpx(no_a(x))   -> no_a(S_cpx(x))
 S_cpx(no_cp(x))  -> no_cp(S_cpx(x))
 S_cpx(no_cp(x))  -> no_cp(S_cpx(S_cpx(x)))
 S_cpx(no_a(x))   -> no_a(x)
 S_cpx(no_cp(x))  -> no_cp(x)
 S_cpx(Answer)    -> Answer
)
```

encodes the forking diagrams and the answer diagram, where `S_cpx` represents $\xleftarrow{S,cpx}$. The first two rules encode the first diagram (there are two cases: the normal order reduction is a (cp)-reduction, or it is not a (cp)-reduction). The third rule encodes the second diagram, the fourth and fifth rule encode the third diagram, and the last rule encodes the answer diagram.

The same TRS encodes the commuting diagrams and the answer diagram, where `S_cpx` represents $\xrightarrow{S,cpx}$

AProVE shows termination of the TRS and thus this shows that for every surface context S and $e_1 \xrightarrow{cpx} e_2$: $S[e_1]{\Downarrow} \iff S[e_2]{\Downarrow}$. Since every surface context is also a reduction context, the context lemma (Lemma 3.15) shows $e_1 \sim_c e_2$ and thus (cpx) is a correct program transformation.

Solution to Exercise 4.9

1. The expression `True` is a WHNF and thus `True`$\downarrow$, `True`$\Downarrow$, and `True`$\downarrow\downarrow$
2. The expression $\lambda x.(\texttt{letrec}\ y = y\ \texttt{in}\ y)$ is also a WHNF and thus it is may-, should-, and must-convergent.
3. The expression `choice True False` non-deterministically reduces in normal order to `True` or to `False`. Since `True` and `False` are WHNFs, all normal order reduction sequences are finite and end in a WHNF, i.e. (`choice True False`) is may-convergent, should-convergent, and must-convergent.
4. The expression $e := \texttt{letrec}\ x = \texttt{choice}\ x\ y, y = \texttt{Nil}\ \texttt{in}\ x$ either reduces in normal order to $\texttt{letrec}\ x = x, y = \texttt{Nil}\ \texttt{in}\ x$ or to $\texttt{letrec}\ x = y, y = \texttt{Nil}\ \texttt{in}\ x$. The former result is a must-divergent expression (since it is irreducible, but not a WHNF), while the latter expression is a WHNF. Thus e is may-convergent, but neither should- nor must-convergent.
5. The expression $\texttt{letrec}\ x = \texttt{choice}\ x\ y, y = \texttt{Nil}\ \texttt{in}\ y$ is a WHNF and thus it is may-, should-, and must-convergent.
6. The expression $e := \texttt{letrec}\ x = \lambda z.(\texttt{choice}\ (x\ \texttt{Nil})\ y), y = \texttt{Nil}\ \texttt{in}\ (x\ \texttt{Nil})$ either reduces in normal order to $e_1 := \texttt{letrec}\ x = \lambda z.(\texttt{choice}\ (x\ \texttt{Nil})\ y), y = \texttt{Nil}, z = \texttt{Nil}\ \texttt{in}\ y$ which is a WHNF or to $e_2 := \texttt{letrec}\ x = \lambda z.(\texttt{choice}\ (x\ \texttt{Nil})\ y), y = \texttt{Nil}, z = \texttt{Nil}\ \texttt{in}\ (x\ \texttt{Nil})$ which is contextually equivalent to e. The reduction $e \xrightarrow{no,*} e_1$ shows that $e\downarrow$. Since $e \sim_c e_2$ we also have that the ability to converge is never lost for every successor w.r.t. normal order reduction, and thus $e\Downarrow$. However, e has an infinite normal order reduction (by always choosing the right argument of `choice`) and thus $\neg(e\downarrow\downarrow)$.