Exercise 1 (Reduction Relation with Closures)

For the evaluation of lambda terms that is closer to evaluation of programs in functional programming languages, one usually replaces textual substitution $t[v/x]$ with a more lazy approach that records the binding $x \mapsto v$ in an environment. These bindings are used whenever needed the value of a variable $v$.

In this approach abstractions $\lambda x.t$ do not evaluate to themselves, but to a pair $(\lambda x.t)[e]$, where $e$ is the current environment. We call such pairs function closures.

a) Define a big-step reduction relation for the lambda calculus with function closures and environments.

b) Add explicit error handling for the case where the binding of a free variable $v$ cannot be found in the environment. Introduce an explicit value abort to indicate such an exception in the relation.

Solution

a)

\[
\frac{e(x) = v}{e \vdash x \Rightarrow_{cbv} v} \quad e \vdash \lambda x.t \Rightarrow_{cbv} (\lambda x.v)[e]
\]

\[
\frac{e \vdash t_1 \Rightarrow_{cbv} (\lambda x.t)[e']}{e \vdash t_2 \Rightarrow_{cbv} v'} \quad e + (x \mapsto v') \vdash t \Rightarrow_{cbv} v
\]

\[
e \vdash t_1 \Rightarrow_{cbv} v
\]

\[
e \vdash t_2 \Rightarrow_{cbv} v
\]

\[
e \vdash t_1 t_2 \Rightarrow_{cbv} abort
\]

b) We just need to add rules to propagate errors, and modify the existing rules to ensure that no subexpression evaluates to abort.

\[
\frac{x \notin e}{e \vdash x \Rightarrow_{cbv} abort}
\]

\[
\frac{e(x) = v}{e \vdash x \Rightarrow_{cbv} v}
\]

\[
\frac{e \vdash \lambda x.t \Rightarrow_{cbv} (\lambda x.v)[e]}{e \vdash \lambda x.t \Rightarrow_{cbv} (\lambda x.v)[e']}
\]

\[
\frac{e \vdash t_1 \Rightarrow_{cbv} (\lambda x.t)[e']}{e \vdash t_2 \Rightarrow_{cbv} v'} \quad e' + (x \mapsto v') \vdash t \Rightarrow_{cbv} v \quad v' \neq abort
\]

\[
e \vdash t_1 t_2 \Rightarrow_{cbv} abort
\]

\[
e \vdash t_1 \Rightarrow_{cbv} abort
\]

\[
e \vdash t_2 \Rightarrow_{cbv} abort
\]

\[
e \vdash t_1 t_2 \Rightarrow_{cbv} abort
\]
Exercise 2 (Reduction Relation with Pattern Matching)

In this exercise, we consider a λ-calculus extended with a special set of constructor values and pattern matching. Constructor values are constructed according to the following grammar:

\[ c ::= C (c_1, \ldots, c_n) \text{ for } n \geq 0 \]

where \( C \) is one from a distinguished set of constructor symbols.

We illustrate pattern matching by example. The expression

\[
\text{match } C_1 \text{ (false) with } C_2 () \rightarrow \text{true} | C_1 (x) \rightarrow x
\]

should evaluate to false, while

\[
\text{match } C_2 \text{ (false) with } C_2 () \rightarrow \text{true} | C_1 (x) \rightarrow x
\]

should evaluate to abort.

a) Define a big-step reduction relation for this language.

b) Prove that the two derivations stated informally above are indeed possible in the relation.

Solution

a) We need two things here: a relation that determines weather a pattern matches a constructor term and produces the corresponding variable bindings; and additional cases in the big step semantics that work through the patterns of a match-construct case by case.

The relation for matching a single pattern can look like this:

\[
e \vdash v \Downarrow x \Rightarrow e + (x \mapsto v)
\]

\[
e \vdash c_1 \Downarrow p_1 \Rightarrow e_1, e_1 \vdash c_2 \Downarrow p_2 \Rightarrow e_2, \ldots
\]

\[
e \vdash C c_1 c_2 \ldots c_n \Downarrow C p_1 p_2 \ldots p_n \Rightarrow e_n
\]

We did not explicitly give the rules that produce and propagate abort in the case of a mismatch.

Now we extend the big-step relation from the last exercise:

\[
e \vdash c_1 \Downarrow p_1 \Rightarrow e_1, e_1 \vdash t_1 \Rightarrow_{cbv} v
\]

\[
e \vdash \text{match } c \text{ with } p_1 \rightarrow t_1 | \ldots \Rightarrow_{cbv} v
\]

\[
e \vdash \text{match } c \text{ with } p_1 \Rightarrow \text{abort}, e \vdash \text{match } c \text{ with } p_2 \rightarrow t_2 | \ldots \Rightarrow_{cbv} v
\]

\[
e \vdash \text{match } c \text{ with } p_1 \rightarrow t_1 | p_2 \rightarrow t_2 | \ldots \Rightarrow_{cbv} v
\]

\[
e \vdash \text{match } c \text{ with } [] \Rightarrow_{cbv} \text{ abort}
\]
• We illustrate the first case only. We first show:

\[
\begin{align*}
\{\} & \vdash \text{false} \Downarrow x \Rightarrow \{x \mapsto \text{false}\} \\
\{\} & \vdash C_1(\text{false}) \Downarrow C_1(x) \Rightarrow \{x \mapsto \text{false}\}
\end{align*}
\]

and

\[
\begin{align*}
\{\} & \vdash C_1(\text{false}) \Downarrow C_2() \Rightarrow \text{abort}.
\end{align*}
\]

Then:

\[
\begin{align*}
\{\} & \vdash C_1(\text{false}) \Downarrow C_1(x) \Rightarrow \{x \mapsto \text{false}\} \quad \{x \mapsto \text{false}\} \vdash x \Rightarrow_{\text{cbv}} \text{false} \\
\quad e & \vdash \text{match } C_1(\text{false}) \text{ with } C_1(x) \rightarrow x \Rightarrow_{\text{cbv}} \text{false} \\
\quad e & \vdash C_1(\text{false}) \Downarrow C_2() \Rightarrow \text{abort} \\
\quad e & \vdash \text{match } C_1(\text{false}) \text{ with } C_2() \rightarrow \text{true} | C_1(x) \rightarrow x \Rightarrow_{\text{cbv}} \text{false}
\end{align*}
\]

**Homework 3 (Normal Forms)**

Recall the inductive definition of the set NF of normal forms:

\[
\begin{align*}
& t \in \text{NF} \quad \lambda x. t \in \text{NF} \\
& n \geq 0 \quad t_1 \in \text{NF} \quad t_2 \in \text{NF} \quad \ldots \quad t_n \in \text{NF} \\
& x \ t_1 \ t_2 \ \ldots \ t_n \in \text{NF}
\end{align*}
\]

Show that this set precisely captures all normal forms, i.e.:

\[
t \in \text{NF} \iff \not \exists t'. t \rightarrow_{\beta} t'
\]

**Homework 4 (Normal Forms & Big Step)**

Show:

\[
t \in \text{NF} \land t \Rightarrow_n u \Longrightarrow u = t
\]

**Homework 5 (Proofs with Small-steps and Big-steps)**

Let \( \omega := \lambda x. xx \) and

\[
t := (\lambda x. (\lambda xy.x) y) (\omega \omega ((\lambda xy.x) y)) .
\]

Prove the following:

a) \( t \Rightarrow_n z \)

b) \( t \rightarrow_{cbv}^3 t \)

c) \( t \nleftrightarrow_{cbv}^+ t \)