Programming Language Theory 1

Exercise Solutions for Untyped Lambda Calculus

陳亮廷 Chen, Liang-Ting

2020 邏輯、語言與計算暑期研習營
Formosan Summer School on Logic, Language, and Computation

1 Statics

1.1 Free variables

1. $\mathbf{FV}(x\ (y\ z)) = \mathbf{FV}(x) \cup \mathbf{FV}(y\ z) = \{x\} \cup (\mathbf{FV}(y) \cup \mathbf{FV}(z)) = \{x\} \cup (\{y\} \cup \{z\}) = \{x\} \cup (\{y, z\}) = \{x, y, z\}$
2. $\mathbf{FV}(\lambda x.\, y) = \mathbf{FV}(y) - \{x\} = \{y\}$
3. $\mathbf{FV}(\lambda x.\, x) = \mathbf{FV}(x) - \{x\} = \{x\} - \{x\} = \emptyset$
4. $\mathbf{FV}(\lambda sz.\, s\ z) = \mathbf{FV}(\lambda z.\, s\ z) - \{s\} = (\mathbf{FV}(s\ z) - \{z\}) - \{s\} = ((\mathbf{FV}(s) \cup \mathbf{FV}(z)) - \{z\}) - \{s\} = ((\{s, z\}) - \{z\}) - \{s\} = \{s\} - \{s\} = \emptyset$
5. $\mathbf{FV}((\lambda x.\, x)\ \lambda y.\, y) = \mathbf{FV}(\lambda x.\, x) \cup \mathbf{FV}(y.\, y) = (\mathbf{FV}(x) - \{x\}) \cup (\mathbf{FV}(y) - \{y\}) = (\{x\} - \{x\}) \cup (\{y\} - \{y\}) = \emptyset \cup \emptyset = \emptyset$

1.2 Bound variables

The set of variables including free and bound variables in a term M is given by

$$\begin{aligned}
\mathbf{Var}(x) &= x \\
\mathbf{Var}(\lambda x.\, M) &= \{x\} \cup \mathbf{Var}(M) \\
\mathbf{Var}(M\ N) &= \mathbf{Var}(M) \cup \mathbf{Var}(N)
\end{aligned}$$

You may want to define **BV** as those variables which are not free as follows.

$$\mathbf{BV}(M) = \mathbf{Var}(M) - \mathbf{FV}(M).$$

However, in this case, $\mathbf{BV}((\lambda x.\, x)\ x) = \emptyset$ contradicts to the bound occurrence of x on the left hand side $\lambda x.\, x$.

To reflect this situation, the set of bound variables should be defined as

$$\begin{aligned}
\mathbf{BV}'(x) &= \emptyset \\
\mathbf{BV}'(\lambda x.\, M) &= \{x\} \cup \mathbf{BV}'(M) \\
\mathbf{BV}'(M\ N) &= \mathbf{BV}'(M) \cup \mathbf{BV}'(N)
\end{aligned}$$

Therefore, we have $x \in \mathbf{BV}'((\lambda x.\, x)\ x)$ and $x \in \mathbf{FV}((\lambda x.\, x)\ x)$. That is, a variable can be free and bound *at the same time*. Yet, it is clear that x occurs in two different positions: one is in the left hand side and another one the right hand side of the application. The ambiguity can be resolved if we introduce the notion of *occurrence*.

1.3 α-equivalence

1. x and y are not α-equivalent, since free variables are α-equivalent if and only if they are the same.

2. $\lambda x\, y.\, y$ and $\lambda z\, y.\, y$ are equivalent, as we have the following derivation:

$$\dfrac{\lambda x.\, \lambda y.\, y \to_\alpha \lambda z.\, (\lambda y.\, y)[z/x]}{\lambda x.\, \lambda y.\, y =_\alpha \lambda z.\, \lambda y.\, y}$$

   where $(\lambda y.\, y)[z/y] \equiv \lambda y.\, y$ by definition.

3. $\lambda x\, y.\, x$ and $\lambda y\, x.\, y$ are α-equivalent by the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\lambda y.\, x \to_\alpha \lambda z.\, x[z/y]}{\lambda y.\, x =_\alpha \lambda z.\, x}}{\lambda x.\, (\lambda y.\, x) =_\alpha \lambda x.\, (\lambda z.\, x)} \quad \dfrac{\lambda x.\, \lambda z.\, x \to_\alpha \lambda y.\, (\lambda z.\, x)[y/x]}{\lambda x.\, \lambda z.\, x =_\alpha \lambda y.\, \lambda z.\, y}}{\lambda x.\, \lambda y.\, x =_\alpha \lambda y.\, \lambda z.\, y}\text{transitivity} \quad \dfrac{\dfrac{\lambda z.\, y \to_\alpha \lambda x.\, y[x/z]}{\lambda z.\, y =_\alpha \lambda x.\, y}}{\lambda y.\, \lambda z.\, y =_\alpha \lambda y.\, \lambda x.\, y}}{\lambda x.\, \lambda y.\, x =_\alpha \lambda y.\, \lambda x.\, y}\text{transitivity}$$

4. $\lambda x\, y.\, x$ and $\lambda x\, y.\, y$ are not α-equivalent, since $\lambda x.\, M_1$ and $\lambda x.\, M_2$ are α-equivalent if and only if M_1 and M_2 are α-equivalent. Similarly, $M_1 = \lambda y.\, x$ and $M_2 = \lambda y.\, y$ are α-equivalent if $x =_\alpha y$ which is not.

2 Dynamics

2.1 α-conversion during β-reduction

We demonstrate the evaluation of the following term by exhibiting the first few reductions.

$$
\begin{aligned}
\underbrace{(\lambda y.\, y\ s\ y)\ (\lambda t\, z\, x. z\ (t\ x)\ z)}_{\beta\text{-redex}} &\longrightarrow_{\beta 1} (y\ s\ y)[(\lambda t\, z\, x.\, z\ (t\ x)\ z)/y] \\
&\equiv \underbrace{(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ s}\ (\lambda t\, z\, x. z\ (t\ x)\ z) \\
&\longrightarrow_{\beta 1} (\lambda z\, x.\, z\ (t\ x)\ z)[s/t]\ (\lambda t\, z\, x. z\ (t\ x)\ z) \\
&\equiv \underbrace{(\lambda z\, x.\, z\ (s\ x)\ z)\ (\lambda t\, z\, x.\, z\ (t\ x)\ z)} \\
&\longrightarrow_{\beta 1} (\lambda x.\, z\ (s\ x)\ z)[(\lambda t\, z\, x.\, z\ (t\ x)\ z)/z] \\
&\equiv \lambda x.\ \underbrace{(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ (s\ x)}\ (\lambda t\, z\, x.\, z\ (t\ x)\ z) \\
&=_\alpha \lambda x.\ \underbrace{(\lambda t\, z\, x'.\, z\ (t\ x')\ z)\ (s\ x)}\ (\lambda t\, z\, x.\, z\ (t\ x)\ z) \\
&\longrightarrow_{\beta 1} \lambda x.\, (\lambda z\, x'.\, z\ (t\ x')\ z)[(s\ x)/t]\ (\lambda t\, z\, x.\, z\ (t\ x)\ z) \\
&\equiv \lambda x.\ \underbrace{(\lambda z\, x'.\, z\ ((s\ x)\ x')\ z)\ (\lambda t\, z\, x.\, z\ (t\ x)\ z)} \\
&\longrightarrow_{\beta 1} \lambda x.\, (\lambda x'.\, z\ ((s\ x)\ x')\ z)[(\lambda t\, z\, x.\, z\ (t\ x)\ z)/z] \\
&\equiv \lambda x.\, \lambda x'.\ \underbrace{(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ ((s\ x)\ x')}\ (\lambda t\, z\, x.\, z\ (t\ x)\ z) \longrightarrow_{\beta 1} \ldots
\end{aligned}
$$

where α-conversion happens in the 4th β-redex because of $x \in \mathbf{FV}(s\ x)$. Note that in the last reduction the argument of the β-reduction has one variable x' which is given by the α-conversion. There will always be an additional variable generated when evaluating $(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ s$, $(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ (s\ x)$, $(\lambda t\, z\, x.\, z\ (t\ x)\ z)\ ((s\ x)\ x')$, ..., so that this term exhausts as many variables as possible. This bizarre term justifies the requirement of the variable set V being infinite.

3 Programming in λ-calculus

1. Let `flip` be $\lambda x\, y\, z.\, x\ z\ y$. By definition,

$$(\lambda x\, y\, z.\, x\ z\ y)\ M\ N\ P \longrightarrow_{\beta 1} (\lambda y\, z.\, M\ z\ y)\ N\ P \longrightarrow_{\beta 1} (\lambda z.\, M\ z\ N)\ P \longrightarrow_{\beta 1} M\ P\ N$$

2. Define `not`, `and`, and `or` as follows.

$$
\begin{aligned}
\texttt{not} &:= \lambda b\, x\, y.\, b\ y\ x \\
\texttt{and} &:= \lambda b_0\, b_1\, x\, y.\, b_0\ (b_1\, x\, y)\ y \\
\texttt{or} &:= \lambda b_0\, b_1\, x\, y.\, b_0\ x\ (b_1\ x\ y)
\end{aligned}
$$

Some reductions are exhibited below:

$$
\begin{aligned}
\texttt{not true} &\longrightarrow_{\beta 1} \lambda x\, y.\, \texttt{true}\ y\ x &&=_\alpha \lambda x'\, y'.\, \texttt{true}\ y'\ x' \longrightarrow_{\beta 1} \lambda x'\, y'.\, (\lambda y.\, y')\ x' &&\longrightarrow_{\beta 1} \lambda x'\, y'.\, y' =_\alpha \texttt{false} \\
\texttt{not false} &\longrightarrow_{\beta 1} \lambda x\, y.\, \texttt{false}\ y\ x &&=_\alpha \lambda x'\, y'.\, \texttt{false}\ y'\ x' \longrightarrow_{\beta 1} \lambda x'\, y'.\, (\lambda y.\, y)\ x' &&\longrightarrow_{\beta 1} \lambda x'\, y'.\, x' =_\alpha \texttt{true}
\end{aligned}
$$

$$
\begin{aligned}
\texttt{and}\ \texttt{true}\ \texttt{true} &\longrightarrow_{\beta 1} (\lambda b_1\, x\, y.\, \texttt{true}\ (b_1\ x\ y)\ y)\ \texttt{true} \\
&\longrightarrow_{\beta 1} \lambda x\, y.\, \texttt{true}\ (\texttt{true}\ x\ y)\ y \\
&\longrightarrow_{\beta 1} \lambda x\, y.\, (\lambda y'.\, \texttt{true}\ x\ y)\ y \\
&\longrightarrow_{\beta 1} \lambda x\, y.\, \texttt{true}\ x\ y \\
&\longrightarrow_{\beta 1} \lambda x\, y.\, (\lambda y.\, x)\ y \\
&\longrightarrow_{\beta 1} \lambda x\, y.\, x \equiv \texttt{true}
\end{aligned}
$$

3. $\texttt{mult} := \lambda n\, m\, f\, z.\, n\ (\texttt{add}\ (m\ f\ x))\ z$

$$
\begin{aligned}
\texttt{mult}\ \mathbf{c}_0\ \mathbf{c}_2 &\longrightarrow_{\beta 1} (\lambda m\, f\, z.\, \mathbf{c}_0\ (\texttt{add}\ (m\ f\ x))\ z)\ \mathbf{c}_2 \\
&\longrightarrow_{\beta 1} (\lambda m\, f\, z.\, (\lambda z.\, z)\ z)\ \mathbf{c}_2 \\
&\longrightarrow_{\beta 1} \lambda f\, z.\, (\lambda z.\, z)\ z \\
&\longrightarrow_{\beta 1} \lambda f\, z.\, z \\
&\equiv \mathbf{c}_0
\end{aligned}
$$