Elements of Programming Languages
Tutorial 2: Substitution and alpha-equivalence
Week 4 (October 12–16, 2015)

Exercises marked $\star$ are more advanced. Please try all unstarred exercises before the tutorial meeting.

This tutorial will use the following language:

$$
\begin{array}{rcll}
e & ::= & n \mid e_1 \oplus e_2 & \mathsf{L}_{\mathsf{Arith}} \\
& \mid & b \mid e_1 == e_2 \mid \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \mathsf{L}_{\mathsf{If}} \\
& \mid & x \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 & \mathsf{L}_{\mathsf{Let}} \\
& \mid & \lambda x{:}\tau.\ e \mid e_1\ e_2 & \mathsf{L}_{\mathsf{Lam}}
\end{array}
$$

and the associated typing and evaluation rules covered in lectures.

1. **Evaluation**

   (a) Write evaluation derivations showing the result value of the following expressions:

   - $(\lambda x{:}\texttt{int}.\ x)\ 1$
   - $(\lambda x{:}\texttt{int}.\ x + 1)\ 42$
   - $((\lambda x{:}\texttt{int} \to \texttt{int}.\ x)\ (\lambda x{:}\texttt{int}.\ x))\ 1$
   - $(\star)$ $(\lambda f{:}\texttt{int} \to \texttt{int}.\ \lambda x{:}\texttt{int}.f\ (f\ x))\ (\lambda x{:}\texttt{int}.\ x + 1)\ 42$

   (b) $(\star)$ In $\mathsf{L}_{\mathsf{Lam}}$, the `let` operation is *definable*: that is, we can transform an expression $\texttt{let}\ x = e_1\ \texttt{in}\ e_2$ to an expression not involving `let` with the same evaluation behavior. Give such an expression, and show that the evaluation rule for `let` can be obtained from other rules.

2. **Typechecking**

   (a) Write Scala terms using anonymous functions (not **`def`**) having the following types, using only variables and applications in the function body (that is, without using constants or primitive operations):

   - `Int => Int`
   - `Int => Boolean => Int`
   - `(Int => Boolean => String) => (Int => Boolean) => (Int => String)`

   (b) Write typing derivations, and identify the result type, for the following closed expressions, or explain why the expression is not typable.

   - $(\lambda x{:}\texttt{int}.\ x)\ 1$
   - $(\lambda x{:}\texttt{int}.\ x + 1)\ 42$
   - $(\lambda x{:}\texttt{int} \to \texttt{int}.\ x)\ (\lambda x{:}\texttt{int}.\ x)$
   - $(\star)$ $(\lambda x{:}\tau.\ x\ x)$

3. **Alpha-equivalence for $\mathsf{L}_{\mathsf{Lam}}$**

   Recall the partial definition of α-equivalence for $\mathsf{L}_{\mathsf{Let}}$ given in lecture 4:

$$\frac{}{v \equiv_\alpha v} \qquad \frac{}{x \equiv_\alpha x} \qquad \frac{e_1 \equiv_\alpha e_1' \quad e_2 \equiv_\alpha e_2'}{e_1 \oplus e_2 \equiv_\alpha e_1' \oplus e_2'}$$

$$\cdots \qquad \frac{e_1 \equiv_\alpha e_1' \quad e_1[z/x] \equiv_\alpha e_2[z/y] \quad z \notin FV(e_2) \cup FV(e_2')}{\texttt{let}\ x = e_1\ \texttt{in}\ e_2 \equiv_\alpha \texttt{let}\ y = e_1'\ \texttt{in}\ e_2'}$$

   (a) Alpha-equivalence can be visualized by drawing an abstract syntax tree with edges linking the "binding" and "bound" occurrences of variables. Draw abstract syntax trees with binding edges in this way for the following terms:

   - $\texttt{let}\ x = 1\ \texttt{in}\ \texttt{let}\ y = 2\ \texttt{in}\ x + y$
   - $\lambda x.\ \lambda y.\ x + y$
   - $\lambda x.\ \lambda x.\ x + x$
   - $\texttt{let}\ x = 1\ \texttt{in}\ \lambda y.\ x + y$

   (b) ($\star$) Write out the missing rules for α-equivalence for the expressions of L_{If} and $\mathsf{L}_{\mathsf{Lam}}$. (Recall that x is bound in e in $\lambda x.\ e$.)

   (c) ($\star$) Which of the following alpha-equivalence relationships hold?

$$\texttt{if true then}\ y\ \texttt{else}\ z \equiv_\alpha y$$
$$\texttt{let}\ x = y\ \texttt{in}\ (\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z) \equiv_\alpha \texttt{let}\ z = y\ \texttt{in}\ (\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z)$$
$$\texttt{let}\ x = 1\ \texttt{in}\ (\texttt{let}\ y = x\ \texttt{in}\ y + y) \equiv_\alpha \texttt{let}\ x = 1\ \texttt{in}\ (\texttt{let}\ x = x\ \texttt{in}\ x + x)$$
$$\lambda x.\lambda y.x\ y \equiv_\alpha \lambda y.\lambda x.y\ x$$
$$\lambda y.x\ y \equiv_\alpha \lambda x.y\ x$$

4. ($\star$) **Capture-avoiding substitution**

   Recall the definition of capture-avoiding substitution for $\mathsf{L}_{\mathsf{Let}}$:

$$\begin{aligned}
n[e/x] &= n \\
x[e/x] &= e \\
y[e/x] &= y \qquad (x \neq y) \\
(e_1 \oplus e_2)[e/x] &= e_1[e/x] \oplus e_2[e/x] \\
(\texttt{let}\ y = e_1\ \texttt{in}\ e_2)[e/x] &= \texttt{let}\ y = e_1[e/x]\ \texttt{in}\ e_2 \qquad (y = x) \\
(\texttt{let}\ y = e_1\ \texttt{in}\ e_2)[e/x] &= \texttt{let}\ y = e_1[e/x]\ \texttt{in}\ e_2[e/x] \qquad (y \notin FV(e))
\end{aligned}$$

   (a) Following the above pattern, extend the capture-avoiding substitution operation to $\mathsf{L}_{\mathsf{Lam}}$.

   (b) Perform the following substitutions, using alpha-renaming as needed to avoid capture:

$$\begin{aligned}
(\lambda y.\ \lambda z.\ ((x + y) + z))[y \times z/x] &= ??? \\
(\texttt{if}\ x == y\ \texttt{then}\ \lambda z.x\ \texttt{else}\ \lambda x.x)[z/x] &= ???
\end{aligned}$$