Elements of Programming Languages
Tutorial 2: Substitution and alpha-equivalence
Week 4 (October 7–11, 2024)

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

This tutorial will use the following language:

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

and the associated typing and evaluation rules covered in lectures.

1. **Evaluation**

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

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

   (b) $(\star)$ In $\mathsf{L}_{\mathsf{Lam}}$, the `let` operation is *definable*: that is, we can transform an expression $\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2$ 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.

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

3. **Alpha-equivalence for $\mathsf{L}_{\mathsf{Lam}}$** In lecture 4, we defined α-equivalence informally in terms of α-conversion. We can define it directly for $\mathsf{L}_{\mathsf{Let}}$ as follows:

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

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

where the last rule amounts to alpha-converting the two `let`-expressions so that they use the same bound name z, and then checking that their components are equivalent.

(a) 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 $\lambda x.\ e$.)

(b) Which of the following alpha-equivalence relationships hold?

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

(c) ($\star$) The binding structure of an expression 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:

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

4. ($\star$) **Naive substitution and variable capture**

In lecture 4, we discussed how to substitute values for variables. In this exercise we consider the following definition of substitution of *expressions* for variables.

$$\begin{array}{rcll}
v[e/x] & = & v & \\
x[e/x] & = & e & \\
y[e/x] & = & y & (x \neq y) \\
(e_1 \oplus e_2)[e/x] & = & e_1[e/x] \oplus e_2[e/x] & \\
(\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)[e/x] & = & \texttt{if}\ e_0[e/x]\ \texttt{then}\ e_1[e/x]\ \texttt{else}\ e_2[e/x] & \\
(\texttt{let}\ y = e_1\ \texttt{in}\ e_2)[e/x] & = & \texttt{let}\ y = e_1[e/x]\ \texttt{in}\ e_0[e/x] & \\
(\lambda y.\ e_0)[e/x] & = & \lambda y.\ e_0[e/x] &
\end{array}$$

Variable capture occurs when a substitution changes the binding structure of an expression. This definition is naive because it permits variable capture.

(a) Perform the following substitutions using the naive definition.

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

(b) Next, α-convert the expressions above to use fresh bound names such as a, b, c, d.

(c) Finally, perform the above substitutions on the α-converted expressions. Are the results α-equivalent?