15-312 Lecture on Substitutions

Here, we will consider the language

$$\begin{array}{llll} \textit{Expressions} & e & ::= & \mathsf{num}[n] \mid \mathsf{plus}(e_1, e_2) \\ & & \mid & \mathsf{str}[s] \mid \mathsf{cat}(e_1, e_2) \\ & & \mid & x \mid \mathsf{let}(e_1, x.e_2) \end{array}$$

but this can be generalized to any language.

We will define what it means to substitute an expression e for a name x in another expression e', which we write $[e/x]e'$. As a concrete example (using concrete syntax), we will need to do this when evaluating an expression of the form

$$\begin{array}{l} \texttt{let} \\ \quad x = e_1 \\ \texttt{in} \\ \quad e_2 \\ \texttt{end} \end{array}$$

Substitutions Assuming Automatic α-Renaming

Assuming terms are automatically α-renamed so that bound are always different from previously encountered names, $[e/x]e'$ is defined inductively on the structure of e' by the following equalities:

$$\begin{array}{lcll} [e/x](\mathsf{num}[n]) & = & \mathsf{num}[n] & \\ [e/x](\mathsf{plus}(e'_1, e'_2)) & = & \mathsf{plus}([e/x]e'_1, [e/x]e'_2) & \\ [e/x](\mathsf{str}[s]) & = & \mathsf{str}[s] & \\ [e/x](\mathsf{cat}(e'_1, e'_2)) & = & \mathsf{cat}([e/x]e'_1, [e/x]e'_2) & \\ [e/x]x & = & e & \\ [e/x]y & = & y & \text{for } y \neq x \\ [e/x](\mathsf{let}(e'_1, z.e'_2)) & = & \mathsf{let}([e/x]e'_1, z.([e/x]e'_2)) & \end{array}$$

Here, automatic α-renaming transparently changes the name of the bound variable of the let to some new name, say z, that does not appear neither in e_2 nor in e. For example, it automatically rewrites

$$\begin{array}{l}\texttt{let}\\ \quad x = 2 * y\\ \texttt{in}\\ \quad \texttt{let}\ x = 2 * x\ \texttt{in}\ x + x\ \texttt{end}\\ \texttt{end}\end{array}$$

to

$$\begin{array}{l}\texttt{let}\\ \quad x = 2 * y\\ \texttt{in}\\ \quad \texttt{let}\ z = 2 * x\ \texttt{in}\ z + z\ \texttt{end}\\ \texttt{end}\end{array}$$

We will always assume to have automatic α-renaming, but let's see what would happen if we didn't have it.

Substitutions without Automatic α-Renaming

The only equalities that need to be change are the one with binders, here the one about let. Let's examine a couple of cases and come up with definitions for them.

Same Variable

Let's consider the last example again. We have

$$\begin{array}{l}\texttt{let}\\ \quad x = 2 * y\\ \texttt{in}\\ \quad \texttt{let}\ x = 2 * x\ \texttt{in}\ x + x\ \texttt{end}\\ \texttt{end}\end{array}$$

and to evaluate it we want to substitute $2 * y$ for x in $\texttt{let}\ x = 2 * x\ \texttt{in}\ x + x\ \texttt{end}$. Then, we are free to carry out the substitution in the subterm $2 * x$ which is bound by the outer x, but we should leave $x + x$ alone because it is bound by the inner x. So there result should be $\texttt{let}\ x = 2 * 2 * y\ \texttt{in}\ x + x\ \texttt{end}$. The general rule is then:

$$[e/x](\mathsf{let}(e_1', x.e_2')) \ = \ \mathsf{let}([e/x]e_1', x.e_2'))$$

All the occurrences of x in e_2' will be bound by the "x." of this $\texttt{let}$, so there are no "free" occurrences of x in $x.e_2'$.

Different Variable — Case 1

Let's consider a small variant of the second example, then:

$$\begin{array}{l}\texttt{let}\\ \quad x = 2 * y\\ \texttt{in}\\ \quad \texttt{let}\ z = 2 * x\ \texttt{in}\ x + z\ \texttt{end}\\ \texttt{end}\end{array}$$

Here, substituting $2 * y$ for x in `let` $z = 2 * x$ `in` $x + z$ `end` is a simple syntactic substitution and the result is `let` $z = 2 * 2 * y$ `in` $2 * y + z$ `end`. The definition in this case seems to be something like this:

$$[e/x](\mathsf{let}(e_1', z.e_2')) \;=\; \mathsf{let}([e/x]e_1', z.([e/x]e_2'))) \quad \text{for } z \neq x$$

This is correct only if z does not occur anywhere in e_1. Consider the following example where it does:

let
  $x = 2 * y$
in
  let $y = 2 * x$ in $x + y$ end
end

If we blindly apply this rule, we obtain

`let` $y = 2 * 2 * y$ `in` $2 * y + y$ `end`

and the occurrence of y in the substituting term $2 * y$ has been captured by the binder once substituted for x in $x + y$.

A simple fix is to add the condition that y does not occur free in e. The updated case is as follows:

$$[e/x](\mathsf{let}(e_1', z.e_2')) \;=\; \mathsf{let}([e/x]e_1', z.([e/x]e_2'))) \quad \text{for } z \neq x \text{ and } z \notin \mathrm{FV}(e)$$

But what if z is free in e?

Different Variable — Case 2

. . . then we have to implement α-renaming. We are going to chose a new name, say $\bar{z}$, substitute it for z inside e_2, and bind the result with it. Here, it is important that $\bar{z}$ be new, that is does not occur free in either e_2 or e. The definition becomes:

$$[e/x](\mathsf{let}(e_1', z.e_2')) \;=\; \mathsf{let}([e/x]e_1', \bar{z}.([e/x]e_2''))) \quad \text{where } \bar{z} \text{ is new and } e_2'' = [\bar{z}/z]e_2$$

We may take this definition as a third case for let when $z \neq x$ but $z \in \mathrm{FV}(e)$.

We can also take it as the single definition of substitution for let because it subsumes the other two cases.

Again, we will always assume that α-renaming happens automatically in the background.