More About Bound Variables

Substitution

Our definition of evaluation is based on the "substitution" of values for free variables within terms.

$$(\lambda x.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \qquad \text{(E-AppAbs)}$$

But what is substitution, exactly? How do we define it?

Substitution

For example, what does

$$(\lambda x.\ x\ (\lambda y.\ x\ y))\ (\lambda x.\ x\ y\ x)$$

reduce to?

Note that this example is not a "complete program" — the whole term is not closed. We are mostly interested in the reduction behavior of closed terms, but reduction of open terms is also important in some contexts:

- program optimization
- alternative reduction strategies such as "full beta-reduction"

Formalizing Substitution

Consider the following definition of substitution:

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

Formalizing Substitution

Consider the following definition of substitution:

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

It substitutes for free and *bound* variables!

$$[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{x}.\ \mathtt{x}) = \lambda \mathtt{x}.\mathtt{y}$$

This is not what we want!

Substitution, take two

$$
\begin{array}{lll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{x}.\mathtt{t}_1) = \lambda \mathtt{x}.\ \mathtt{t}_1 & & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) & &
\end{array}
$$

What is wrong with this definition?

Substitution, take two

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{x}.\mathtt{t}_1) = \lambda \mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

It suffers from *variable capture*!

$$[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{y}.\mathtt{x}) = \lambda \mathtt{x}.\ \mathtt{x}$$

This is also not what we want.

Substitution, take three

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin \mathit{FV}(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{x}.\mathtt{t}_1) = \lambda \mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

Substitution, take three

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin \mathit{FV}(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{x}.\mathtt{t}_1) = \lambda \mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

What is wrong with this definition?

Now substition is a *partial function!*

E.g., $[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{y}.\mathtt{x})$ is undefined.

But we want an result for every substitution.

Bound variable names shouldn't matter

It's annoying that that the "spelling" of bound variable names is causing trouble with our definition of substitution.

Intuition tells us that there shouldn't be a difference between the functions $\lambda x.x$ and $\lambda y.y$. Both of these functions do exactly the same thing.

Because they differ only in the names of their bound variables, we'd like to think that these *are* the same function.

We call such terms *alpha-equivalent*.

Alpha-equivalence classes

In fact, we can create equivalence classes of terms that differ only in the names of bound variables.

When working with the lambda calculus, it is convenient to think about these *equivalence classes*, instead of raw terms.

For example, when we write $\lambda \mathtt{x}.\mathtt{x}$ we mean not just this term, but the class of terms that includes $\lambda \mathtt{y}.\mathtt{y}$ and $\lambda \mathtt{z}.\mathtt{z}$.

We can now freely choose a different *representative* from a term's alpha-equivalence class, whenever we need to, to avoid getting stuck.

Substitution, for alpha-equivalence classes

Now consider substitution as an operation over *alpha-equivalence classes* of terms.

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{x} \neq \mathtt{y} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y}.\mathtt{t}_1) = \lambda \mathtt{y}.\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1) & \text{if } \mathtt{x} \neq \mathtt{y},\ \mathtt{y} \notin \mathit{FV}(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{x}.\mathtt{t}_1) = \lambda \mathtt{x}.\ \mathtt{t}_1 & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) &
\end{array}
$$

Examples:

- $[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{y}.\mathtt{x})$ must give the same result as $[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{z}.\mathtt{x})$. We know the latter is $\lambda \mathtt{z}.\mathtt{y}$, so that is what we will use for the former.
- $[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{x}.\mathtt{z})$ must give the same result as $[\mathtt{x} \mapsto \mathtt{y}](\lambda \mathtt{w}.\mathtt{z})$. We know the latter is $\lambda \mathtt{w}.\mathtt{z}$ so that is what we use for the former.

Review

So what does

$$(\lambda x.\ x\ (\lambda y.\ x\ y))\ (\lambda x.\ x\ y\ x)$$

reduce to?