The following is a big-step semantics for the lambda calculus that uses substitution to handle variables.

\[
\begin{align*}
(\lambda x. e) & \Downarrow (\lambda x. e) \\
\Downarrow (\lambda x. e_3) & \quad e_2 \Downarrow v_1 & \quad [x \mapsto v_1]e_3 \Downarrow v_2 \\
(e_1 \ e_2) & \Downarrow v_2
\end{align*}
\]
Another approach to handling variables is to use an environment: a mapping from variables to values. This approach introduces a new kind of value: a closure, which is a $\lambda$ expression together with an environment. The notation $E(x \rightarrow v)$ updates the environment $E$ so that $x$ maps to $v$.

$$E \vdash (\lambda x. e) \downarrow \langle \lambda x. e, E \rangle$$

$$E \vdash x \downarrow E(x)$$

$$E \vdash e_1 \downarrow \langle \lambda x. e_3, E' \rangle \quad E \vdash e_2 \downarrow v_1 \quad E'(x \rightarrow v_1) \vdash e_3 \downarrow v_2$$

$$E \vdash (e_1 e_2) \downarrow v_2$$
Variable Representations

- Motivation: a disadvantage of using symbols for variables is that \( \lambda \) that are equivalent aren’t necessarily syntactically identical:

\[
(\lambda x.x) = (\lambda y.y) = (\lambda z.z)
\]

- One solution is to represent variables as pointers.

\[
(\lambda x.x) = \lambda
\]

\[
(\lambda x. (\lambda y. (x y))) = \lambda
\]
DeBruijn Indices

Another solution is to represent variables with relative addresses called **DeBruijn indices**.

Syntax for the lambda calculus: $e ::= k \mid \lambda. \, e \mid (\, e \, e)$

We use an integer that says which enclosing $\lambda$ is the binding $\lambda$. For example:

$$(\lambda x. \, (\lambda y. \, (x \, y))) = \lambda \quad = (\lambda. \, (\lambda. \, (1 \, 0)))$$
DeBruijn Indices

When we perform substitution \([x \mapsto e']e\), we need to make sure that the indices in \(e'\) still point to the right place, even though we are putting copies of \(e'\) further down in the AST.

So we define a shift operator \(\uparrow^d_c\) that shifts the indices by \(d\) with a cutoff of \(c\). The cutoff is so that when we shift under a \(\lambda\), we don’t shift the variables bound by the \(\lambda\).

\[
\begin{align*}
\uparrow^d_c k &= \text{if } k < c \text{ then } k \text{ else } k + d \\
\uparrow^d_c (\lambda x. \ e) &= \lambda. \uparrow^d_{c+1} e \\
\uparrow^d_c (e_1 \ e_2) &= (\uparrow^d_c e_1 \ \uparrow^d_c e_2)
\end{align*}
\]
Substitution with DeBruijn Indices

- **Definition of substitution:**

\[
[j \mapsto e']k = \text{if } k = j \text{ then } e' \text{ else } k
\]

\[
[j \mapsto e'](\lambda. e) = \lambda. [j + 1 \mapsto 1_0 e']e
\]

\[
[j \mapsto e'](e_1 e_2) = ([j \mapsto e']e_1 [j \mapsto e']e_2)
\]

- **The \(\beta\)-reduction rule:**

\[
(\lambda. e) v \longrightarrow^{\beta \!-1}_0 [0 \mapsto 1_0 v]e
\]

- **Example:**

\[
(\lambda x. ((\lambda y. x) z)) \longrightarrow (\lambda x. [y \mapsto z]x) \longrightarrow (\lambda x. x)
\]

\[
(\lambda. ((\lambda. 1) 1)) \longrightarrow (\lambda. \uparrow^{-1}_0 [0 \mapsto 1_0 1]1) \longrightarrow (\lambda. 0)
\]
The problematic example that had free-variable capture is no problem here:

\[
(\lambda y. ((\lambda x. (\lambda y. x)) y)) \xrightarrow{} (\lambda. ((\lambda. \lambda. 1) 0)) \\
\downarrow \downarrow \downarrow \downarrow \\
(\lambda y. [x \mapsto y](\lambda y. x))) \xrightarrow{} (\lambda. \uparrow^{-1}_{0} [0 \mapsto 1](\lambda. 1)) \\
\downarrow \downarrow \downarrow \downarrow \\
(\lambda y. (\lambda z. [x \mapsto y]x)) \xrightarrow{} (\lambda. \uparrow^{-1}_{0} (\lambda. [1 \mapsto 2]1)) \\
\downarrow \downarrow \downarrow \downarrow \\
(\lambda y. (\lambda z. y)) \xrightarrow{} (\lambda. \uparrow^{-1}_{0} (\lambda. 2)) \\
\downarrow \downarrow \downarrow \downarrow \\
(\lambda y. (\lambda z. y)) \xrightarrow{} (\lambda. (\lambda. \uparrow^{-1}_{1} 2)) \\
\downarrow \downarrow \downarrow \downarrow \\
(\lambda y. (\lambda z. y)) \xrightarrow{} (\lambda. (\lambda. 1))
\]
Locally Nameless Representation

- Motivation: the shifting business for DeBruijn indices is messy and makes it hard for humans to read the lemmas and theorems.
- A new alternative is a hybrid approach that uses DeBruijn indices for bound variables and symbols for free variables.
- Syntax for the lambda calculus: \( e ::= x \mid k \mid (\lambda.\ e) \mid (e\ e) \)
- Substitution for bound variables:

\[
\begin{align*}
\{ k \mapsto u \} i &= (\text{if } k = i \text{ then } u \text{ else } i) \\
\{ k \mapsto u \} x &= x \\
\{ k \mapsto u \} (\lambda.\ e) &= (\lambda.\ \{ k + 1 \mapsto u \} e) \\
\{ k \mapsto u \} (e_1\ e_2) &= (\{ k \mapsto u \} e_1\ k \mapsto u e_2)
\end{align*}
\]
Substitution for free variables:

\[
[x \mapsto e']i = i \\
[x \mapsto e']y = (if \ y = x \ then \ e' \ else \ y) \\
[x \mapsto e'](\lambda. \ e) = (\lambda. \ [x \mapsto e']e) \\
[x \mapsto e'](e_1 e_2) = ([x \mapsto e']e_1 \ [x \mapsto e']e_2)
\]

β-reduction rule:

[((\lambda. \ e) \ v) \rightarrow \{0 \mapsto v\}e