Big-step Semantics

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

$$\frac{}{(\lambda x.\ e) \Downarrow (\lambda x.\ e)}$$

$$\frac{e_1 \Downarrow (\lambda x.\ e_3) \qquad e_2 \Downarrow v_1 \qquad [x \mapsto v_1]e_3 \Downarrow v_2}{(e_1\ e_2) \Downarrow v2}$$

Big-step Semantics

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 λ expression together with an environment. The notation $E(x \rightarrow v)$ updates the environment E so that x maps to v.

$$\frac{}{E \vdash (\lambda x.\ e) \Downarrow \langle \lambda x.\ e, E \rangle}$$

$$\frac{}{E \vdash x \Downarrow E(x)}$$

$$\frac{E \vdash e_1 \Downarrow \langle \lambda x.\ e_3, E' \rangle \qquad E \vdash e_2 \Downarrow v_1 \qquad E'(x \rightarrow v_1) \vdash e_3 \Downarrow v_2}{E \vdash (e_1\ e_2) \Downarrow v_2}$$

Variable Representations

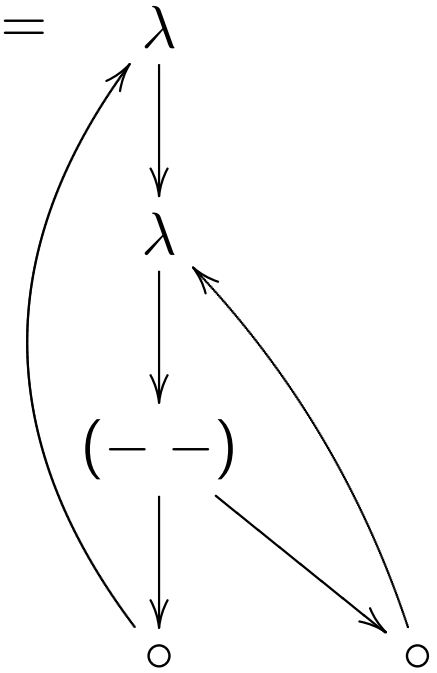
- Motivation: a disadvantage of using symbols for variables is that λ that are equivalent aren't necessarily syntactically identical:

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

- One solution is to represent variables as pointers.

$(\lambda x.x) =$

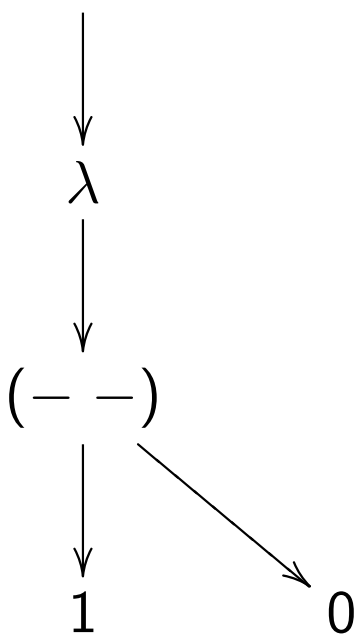
$(\lambda x.\ (\lambda y.\ (x\ y)) =$



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 λ is the binding λ. For example:

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



$= (\lambda.\ (\lambda.\ (1\ 0)))$

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 $\uparrow_c^d$ that shifts the indices by d with a cutoff of c. The cutoff is so that when we shift under a λ, we don't shift the variables bound by the λ.

$$
\begin{aligned}
\uparrow_c^d k &= \mathit{if}\ k < c\ \mathit{then}\ k\ \mathit{else}\ k + d \\
\uparrow_c^d (\lambda x.\ e) &= \lambda.\ \uparrow_{c+1}^d e \\
\uparrow_c^d (e_1\ e_2) &= (\uparrow_c^d e_1\ \uparrow_c^d e_2)
\end{aligned}
$$

Substitution with DeBruijn Indices

- Definition of substitution:

$$\begin{array}{ll} [j \mapsto e']k & = \mathit{if}\ k = j\ \mathit{then}\ e'\ \mathit{else}\ k \\ [j \mapsto e'](\lambda.\ e) & = \lambda.\ [j+1 \mapsto \uparrow_0^1 e']e \\ [j \mapsto e'](e_1\ e_2) & = ([j \mapsto e']e_1\ [j \mapsto e']e_2) \end{array}$$

- The β-reduction rule:

$$(\lambda.\ e)\ v \longrightarrow \uparrow_0^{-1} [0 \mapsto \uparrow_0^1 v]e$$

- Example:

$$\begin{array}{ccccc} (\lambda x.\ ((\lambda y.x)\ z)) & \longrightarrow & (\lambda x.\ [y \mapsto z]x) & \longrightarrow & (\lambda x.\ x) \\ \| & & & & \| \\ (\lambda.\ ((\lambda.\ 1)\ 1)) & \longrightarrow & (\lambda.\ \uparrow_0^{-1} [0 \mapsto \uparrow_0^1 1]1) & \longrightarrow & (\lambda.\ 0) \end{array}$$

Substitution with DeBruijn Indices, continued

The problematic example that had free-variable capture is no problem here:

$$
\begin{array}{ccc}
(\lambda y.\ ((\lambda x.\ (\lambda y.\ x))\ y)) & = & (\lambda.\ ((\lambda.\ \lambda.\ 1)\ 0)) \\
\downarrow & & \downarrow \\
(\lambda y.\ [x \mapsto y](\lambda y.\ x))) & & (\lambda.\ \uparrow_0^{-1} [0 \mapsto 1](\lambda.\ 1)) \\
\downarrow & & \downarrow \\
(\lambda y.\ (\lambda z.\ [x \mapsto y]x)) & & (\lambda.\ \uparrow_0^{-1} (\lambda.\ [1 \mapsto 2]1)) \\
\downarrow & & \downarrow \\
(\lambda y.\ (\lambda z.\ y)) & & (\lambda.\ \uparrow_0^{-1} (\lambda.\ 2)) \\
\downarrow & & \downarrow \\
(\lambda y.\ (\lambda z.\ y)) & & (\lambda.\ (\lambda.\ \uparrow_1^{-1} 2)) \\
\downarrow & & \downarrow \\
(\lambda y.\ (\lambda z.\ y)) & = & (\lambda.\ (\lambda.\ 1))
\end{array}
$$

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:

$$
\begin{aligned}
\{k \mapsto u\}i &= (\mathit{if}\ k = i\ \mathit{then}\ u\ \mathit{else}\ i) \\
\{k \mapsto u\}x &= x \\
\{k \mapsto u\}(\lambda.\ e) &= (\lambda.\ \{k+1 \mapsto u\}e) \\
\{k \mapsto u\}(e_1\ e_2) &= (\{k \mapsto u\}e_1\ k \mapsto ue_2)
\end{aligned}
$$

Locally Nameless Representation

- Substitution for free variables:

$$\begin{array}{ll} [x \mapsto e']i & = i \\ [x \mapsto e']y & = (\mathit{if}\ y = x\ \mathit{then}\ e'\ \mathit{else}\ y) \\ [x \mapsto e'](\lambda.\ e) & = (\lambda.\ [x \mapsto e']e) \\ [x \mapsto e'](e_1\ e_2) & = ([x \mapsto e']e_1\ [x \mapsto e']e_2) \end{array}$$

- β-reduction rule:

$$((\lambda.\ e)\ v) \longrightarrow \{0 \mapsto v\}e$$