de Bruijn Notation

Another way to avoid the tricky issues with substitution is to use a *nameless* representation of terms.

\[ e ::= n \mid \lambda e \mid e e \]
Another way to avoid the tricky issues with substitution is to use a *nameless* representation of terms.

\[ e ::= n \mid \lambda.e \mid e\,e \]

Abstractions have lost their variables!

Variables are replaced with numerical indices!
Examples

Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda x. x )</td>
<td>( \lambda. 0 )</td>
</tr>
<tr>
<td>( \lambda z. z )</td>
<td></td>
</tr>
</tbody>
</table>
Examples

Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda x. \ x$</td>
<td>$\lambda. \ 0$</td>
</tr>
<tr>
<td>$\lambda z. \ z$</td>
<td>$\lambda. \ 0$</td>
</tr>
<tr>
<td>$\lambda x. \ \lambda y. \ x$</td>
<td></td>
</tr>
</tbody>
</table>

Examples

Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda x. x )</td>
<td>( \lambda. 0 )</td>
</tr>
<tr>
<td>( \lambda z. z )</td>
<td>( \lambda. 0 )</td>
</tr>
<tr>
<td>( \lambda x. \lambda y. x )</td>
<td>( \lambda. \lambda. 1 )</td>
</tr>
<tr>
<td>( \lambda x. \lambda y. \lambda s. \lambda z. x s (y s z) )</td>
<td></td>
</tr>
</tbody>
</table>
Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda x. x )</td>
<td>( \lambda. 0 )</td>
</tr>
<tr>
<td>( \lambda z. z )</td>
<td>( \lambda. 0 )</td>
</tr>
<tr>
<td>( \lambda x. \lambda y. x )</td>
<td>( \lambda. \lambda. 1 )</td>
</tr>
<tr>
<td>( \lambda x. \lambda y. \lambda s. \lambda z. x s (y s z) )</td>
<td>( \lambda. \lambda. \lambda. \lambda. 3 1 (2 1 0) )</td>
</tr>
<tr>
<td>( (\lambda x. xx) (\lambda x. xx) )</td>
<td></td>
</tr>
</tbody>
</table>
Examples

Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda x. \ x$</td>
<td>$\lambda. \ 0$</td>
</tr>
<tr>
<td>$\lambda z. \ z$</td>
<td>$\lambda. \ 0$</td>
</tr>
<tr>
<td>$\lambda x. \ \lambda y. \ x$</td>
<td>$\lambda. \ \lambda. \ 1$</td>
</tr>
<tr>
<td>$\lambda x. \ \lambda y. \ \lambda s. \ \lambda z. \ x \ s \ (y \ s \ z)$</td>
<td>$\lambda. \ \lambda. \ \lambda. \ \lambda. \ 3 \ 1 \ (2 \ 1 \ 0)$</td>
</tr>
<tr>
<td>$(\lambda x. \ x \ x) \ (\lambda x. \ x \ x)$</td>
<td>$(\lambda. \ 0 \ 0) \ (\lambda. \ 0 \ 0)$</td>
</tr>
<tr>
<td>$(\lambda x. \ \lambda x. \ x) \ (\lambda y. \ y)$</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
Here are some terms written in standard and de Bruijn notation:

<table>
<thead>
<tr>
<th>Standard</th>
<th>de Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda x. x$</td>
<td>$\lambda. 0$</td>
</tr>
<tr>
<td>$\lambda z. z$</td>
<td>$\lambda. 0$</td>
</tr>
<tr>
<td>$\lambda x. \lambda y. x$</td>
<td>$\lambda. \lambda. 1$</td>
</tr>
<tr>
<td>$\lambda x. \lambda y. \lambda s. \lambda z. x s (y s z)$</td>
<td>$\lambda. \lambda. \lambda. \lambda. 31(210)$</td>
</tr>
<tr>
<td>$(\lambda x. x x) (\lambda x. x x)$</td>
<td>$(\lambda. 00) (\lambda. 00)$</td>
</tr>
<tr>
<td>$(\lambda x. \lambda x. x) (\lambda y. y)$</td>
<td>$(\lambda. \lambda. 0) (\lambda. 0)$</td>
</tr>
</tbody>
</table>
Free variables

To represent a $\lambda$-expression that contains free variables in de Bruijn notation, we need a way to map the free variables to integers.

We will work with respect to a map $\Gamma$ from variables to integers called a context.

Examples:

Suppose that $\Gamma$ maps $x$ to 0 and $y$ to 1.

- Representation of $xy$ is $01$
- Representation of $\lambda z. x y z \lambda$. $120$
Shifting

To define substitution, we will need an operation that shifts by \( i \) the variables above a cutoff \( c \):

\[
\uparrow^i_c (n) = \begin{cases} 
  n & \text{if } n < c \\
  n + i & \text{otherwise}
\end{cases}
\]

\[
\uparrow^i_c (\lambda.e) = \lambda.(\uparrow^i_{c+1} e)
\]

\[
\uparrow^i_c (e_1 e_2) = (\uparrow^i_c e_1) (\uparrow^i_c e_2)
\]

The cutoff \( c \) keeps track of the variables that were bound in the original expression and so should not be shifted.

The cutoff is 0 initially.
Substitution

Now we can define substitution:

\[
\begin{align*}
  n\{e/m\} &= \begin{cases} 
e & \text{if } n = m \\ 
n & \text{otherwise} \end{cases} \\
  (\lambda e_1)\{e/m\} &= \lambda e_1\{e_1\{e/m\}/m + 1\} \\
  (e_1 e_2)\{e/m\} &= (e_1\{e/m\})(e_2\{e/m\})
\end{align*}
\]
Substitution

Now we can define substitution:

\[
\begin{align*}
n\{e/m\} & = \begin{cases} 
e & \text{if } n = m \\ n & \text{otherwise} \end{cases} \\
(\lambda.e_1)\{e/m\} & = \lambda.e_1\{e/0\}/m + 1 \\
(e_1 e_2)\{e/m\} & = (e_1\{e/m\}) (e_2\{e/m\})
\end{align*}
\]

The $\beta$ rule for terms in de Bruijn notation is just:

\[
\begin{align*}
(\lambda.e_1) e_2 & \rightarrow \uparrow_0^{-1} (e_1\{\uparrow_0^1 e_2/0\}) \\
\end{align*}
\]
Consider the term \((\lambda u.\lambda v. u\; x)\; y\) with respect to a context where \\
\(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[(\lambda. \lambda.1 \ 2) \ 1\]
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda.\lambda.1\ 2)\ 1 \\
\to \uparrow_{0}^{-1} ((\lambda.1\ 2)((\uparrow_{0}^{1} 1)/0))
\]
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda. \lambda. 1 \ 2) \ 1 \\
\rightarrow \ \uparrow_0^{-1} ( (\lambda. 1 \ 2)\{ (\uparrow_0^{-1} 1)/0 \} ) \\
= \ \uparrow_0^{-1} ( (\lambda. 1 \ 2)\{2/0 \} )
\]
Example

Consider the term \((\lambda u.\lambda v. u\ x)\ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
\begin{align*}
(\lambda.\lambda.1\ 2)\ 1 \\
\rightarrow & \uparrow^{-1}_0 (((\lambda.1\ 2)\{(\uparrow^1_0\ 1)/0\})) \\
= & \uparrow^{-1}_0 ((\lambda.1\ 2)\{2/0\}) \\
= & \uparrow^{-1}_0 \lambda.((1\ 2)\{(\uparrow^1_0\ 2)/(0 + 1)\})
\end{align*}
\]
Example

Consider the term \((\lambda u.\lambda v.u\ x)\ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda.\lambda.1\ 2)\ 1 \\
\rightarrow \uparrow_0^{-1} (((\lambda.1\ 2)\{(\uparrow_0^1 1)/0\})) \\
= \uparrow_0^{-1} (((\lambda.1\ 2)\{2/0\})) \\
= \uparrow_0^{-1} \lambda.((1\ 2)\{(\uparrow_0^1 2)/(0 + 1)\}) \\
= \uparrow_0^{-1} \lambda.((1\ 2)\{3/1\})
\]
Example

Consider the term \((\lambda u. \lambda v. u x) \, y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda. \lambda. 1\, 2) \, 1 \\
\rightarrow \uparrow^{-1}_0 (((\lambda. 1\, 2)\{((\uparrow^{1}_0 1)/0\}) \\
= \uparrow^{-1}_0 (((\lambda. 1\, 2)\{2/0\}) \\
= \uparrow^{-1}_0 \lambda.((1\, 2)\{((\uparrow^{1}_0 2)/(0 + 1))\}) \\
= \uparrow^{-1}_0 \lambda.((1\, 2)\{3/1\}) \\
= \uparrow^{-1}_0 \lambda.(1\{3/1\}) (2\{3/1\})
\]
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda. \lambda.1\ 2)\ 1 \\
\rightarrow \uparrow^{-1}_0 ((\lambda.1\ 2)\{(\uparrow^1_0 1)/0\}) \\
= \uparrow^{-1}_0 ((\lambda.1\ 2)\{2/0\}) \\
= \uparrow^{-1}_0 \lambda.((1\ 2)\{(\uparrow^1_0 2)/(0 + 1)\}) \\
= \uparrow^{-1}_0 \lambda.((1\ 2)\{3/1\}) \\
= \uparrow^{-1}_0 \lambda.(1\{3/1\})\ (2\{3/1\}) \\
= \uparrow^{-1}_0 \lambda.3\ 2
\]
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
(\lambda. \lambda.1\ 2)\ 1 \\
\rightarrow \ \uparrow^{-1}_{0} ((\lambda.1\ 2)\{\uparrow^{-1}_{0} 1/0\}) \\
= \ \uparrow^{-1}_{0} ((\lambda.1\ 2)\{2/0\}) \\
= \ \uparrow^{-1}_{0} \lambda.((1\ 2)\{\uparrow^{-1}_{0} 2/(0 + 1)\}) \\
= \ \uparrow^{-1}_{0} \lambda.((1\ 2)\{3/1\}) \\
= \ \uparrow^{-1}_{0} \lambda.(1\{3/1\})\ (2\{3/1\}) \\
= \ \uparrow^{-1}_{0} \lambda.3\ 2 \\
= \ \lambda.2\ 1
\]
Example

Consider the term \((\lambda u. \lambda v. u \ x) \ y\) with respect to a context where \(\Gamma(x) = 0\) and \(\Gamma(y) = 1\).

\[
\begin{align*}
(\lambda. \lambda.1\ 2)\ 1 & \\
\Rightarrow & \ \uparrow^{-1}_0 (((\lambda.1\ 2)\{((\uparrow^1_0\ 1)/0\})
\Rightarrow & \ \uparrow^{-1}_0 (((\lambda.1\ 2)\{2/0\})
\Rightarrow & \ \uparrow^{-1}_0 \lambda.((1\ 2)\{(\uparrow^1_0\ 2)/(0 + 1)\})
\Rightarrow & \ \uparrow^{-1}_0 \lambda.((1\ 2)\{3/1\})
\Rightarrow & \ \uparrow^{-1}_0 \lambda.(1\{3/1\})\ (2\{3/1\})
\Rightarrow & \ \uparrow^{-1}_0 \lambda.3\ 2
\Rightarrow & \ \lambda.2\ 1
\end{align*}
\]

which, in standard notation (with respect to \(\Gamma\)), is the same as \(\lambda v. y \ x\).
Combinators

Another way to avoid the issues having to do with free and bound variable names in the $\lambda$-calculus is to work with closed expressions or *combinators*.

With just three combinators, we can encode the entire $\lambda$-calculus.

\[
\begin{align*}
K &= \lambda x. \lambda y. x \\
S &= \lambda x. \lambda y. \lambda z. xz(yz) \\
I &= \lambda x. x
\end{align*}
\]
Combinators

Another way to avoid the issues having to do with free and bound variable names in the λ-calculus is to work with closed expressions or *combinators*.

With just three combinators, we can encode the entire λ-calculus.

\[
\begin{align*}
K &= \lambda x. \lambda y. x \\
S &= \lambda x. \lambda y. \lambda z. x \, z \, (y \, z) \\
I &= \lambda x. x
\end{align*}
\]
We can even define independent evaluation rules that don’t depend on the $\lambda$-calculus at all.

Behold the “SKI-calculus”:

$$K \, e_1 \, e_2 \rightarrow e_1$$
$$S \, e_1 \, e_2 \, e_3 \rightarrow e_1 \, e_3 \, (e_2 \, e_3)$$
$$I \, e \rightarrow e$$

You would never want to program in this language—it doesn’t even have variables!—but it’s exactly as powerful as the $\lambda$-calculus.
Bracket Abstraction

The function \([x]\) that takes a combinator term \(M\) and builds another term that behaves like \(\lambda x.M\):

\[
\begin{align*}
[x] x & = I \\
[x] N & = K N \\
[x] N_1 N_2 & = S ([x] N_1) ([x] N_2)
\end{align*}
\]

where \(x \not\in \text{fv}(N)\)

The idea is that \(([x] M) N \rightarrow M\{N/x\}\) for every term \(N\).
Bracket Abstraction

We then define a function \((e)\ast\) that maps a \(\lambda\)-calculus expression to a combinator term:

\[
\begin{align*}
(x)\ast & = x \\
(e_1 e_2)\ast & = (e_1)\ast (e_2)\ast \\
(\lambda x. e)\ast & = [x] (e)\ast
\end{align*}
\]
As an example, the expression $\lambda x. \lambda y. x$ is translated as follows:

$$(\lambda x. \lambda y. x)^*$$  
$$= [x] (\lambda y. x)^*$$  
$$= [x] ([y] x)$$  
$$= [x] (K x)$$  
$$= (S ([x] K) ([x] x))$$  
$$= S (K K) I$$

No variables in the final combinator term!
Example

We can check that this behaves the same as our original \( \lambda \)-expression by seeing how it evaluates when applied to arbitrary expressions \( e_1 \) and \( e_2 \).

\[
(\lambda x. \lambda y. x) \ e_1 \ e_2 \\
\rightarrow \ (\lambda y. \ e_1) \ e_2 \\
\rightarrow \ e_1
\]
Example

We can check that this behaves the same as our original \( \lambda \)-expression by seeing how it evaluates when applied to arbitrary expressions \( e_1 \) and \( e_2 \).

\[
(\lambda x. \lambda y. x) \ e_1 \ e_2 \\
\rightarrow (\lambda y. e_1) \ e_2 \\
\rightarrow e_1
\]

and

\[
(S \ (K \ K) \ I) \ e_1 \ e_2 \\
\rightarrow (K \ K \ e_1) \ (I \ e_1) \ e_2 \\
\rightarrow K \ e_1 \ e_2 \\
\rightarrow e_1
\]
Looking back at our definitions...

\[
\begin{align*}
K & e_1 e_2 \rightarrow e_1 \\
S & e_1 e_2 e_3 \rightarrow e_1 e_3 (e_2 e_3) \\
I & e \rightarrow e
\end{align*}
\]

...I isn’t strictly necessary. It behaves the same as S K K.
Looking back at our definitions...

\[ K \ e_1 \ e_2 \rightarrow e_1 \]
\[ S \ e_1 \ e_2 \ e_3 \rightarrow e_1 \ e_3 \ (e_2 \ e_3) \]
\[ I \ e \rightarrow e \]

...I isn’t strictly necessary. It behaves the same as S K K.

Our example becomes:

\[ S \ (K \ K) \ (S \ K \ K) \]
One Step Farther

If two combinators are enough, how about one?

\[ \iota \equiv \lambda f. S K \]

Then:

\[ I = \beta \iota \iota \]
\[ K = \beta \iota (\iota (\iota \iota)) \]
\[ S = \beta \iota (\iota (\iota (\iota \iota))) \]

In this “language,” programs only differ in the shape of the tree!
One Step Farther

If two combinators are enough, how about one?

\[ \iota \triangleq \lambda f. f \ S \ K \]
If two combinators are enough, how about one?

\[ \iota \triangleq \lambda f. f \ S \ K \]

Then:

\[
\begin{align*}
I &= \beta \iota \\
K &= \beta \iota(\iota(\iota)) \\
S &= \beta \iota(\iota(\iota(\iota)))
\end{align*}
\]

In this “language,” programs only differ in the shape of the tree!