CS 4110

Programming Languages & Logics

Lecture 17
De Bruijn, Combinators

de Bruijn Notation

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

$$e ::= n \mid \lambda.e \mid e\,e$$

de Bruijn Notation

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

$$e ::= n \mid \lambda.e \mid e\ e$$

Abstractions have lost their variables!

Variables are replaced with numerical indices!

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	$\lambda.\ 0$
$\lambda x.\ \lambda y.\ x$	

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	$\lambda.\ 0$
$\lambda x.\ \lambda y.\ x$	$\lambda.\ \lambda.\ 1$
$\lambda x.\ \lambda y.\ \lambda s.\ \lambda z.\ x\, s\, (y\, s\, z)$	

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	$\lambda.\ 0$
$\lambda x.\ \lambda y.\ x$	$\lambda.\ \lambda.\ 1$
$\lambda x.\ \lambda y.\ \lambda s.\ \lambda z.\ x\ s\ (y\ s\ z)$	$\lambda.\ \lambda.\ \lambda.\ \lambda.\ 3\ 1\ (2\ 1\ 0)$
$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$	

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	$\lambda.\ 0$
$\lambda x.\ \lambda y.\ x$	$\lambda.\ \lambda.\ 1$
$\lambda x.\ \lambda y.\ \lambda s.\ \lambda z.\ x\ s\ (y\ s\ z)$	$\lambda.\ \lambda.\ \lambda.\ \lambda.\ 3\ 1\ (2\ 1\ 0)$
$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$	$(\lambda.\ 0\ 0)\ (\lambda.\ 0\ 0)$
$(\lambda x.\ \lambda x.\ x)\ (\lambda y.\ y)$	

Examples

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

Standard	**de Bruijn**
$\lambda x.\ x$	$\lambda.\ 0$
$\lambda z.\ z$	$\lambda.\ 0$
$\lambda x.\ \lambda y.\ x$	$\lambda.\ \lambda.\ 1$
$\lambda x.\ \lambda y.\ \lambda s.\ \lambda z.\ x\ s\ (y\ s\ z)$	$\lambda.\ \lambda.\ \lambda.\ \lambda.\ 3\ 1\ (2\ 1\ 0)$
$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$	$(\lambda.\ 0\ 0)\ (\lambda.\ 0\ 0)$
$(\lambda x.\ \lambda x.\ x)\ (\lambda y.\ y)$	$(\lambda.\ \lambda.\ 0)\ (\lambda.\ 0)$

Free variables

To represent a λ-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 Γ from variables to integers called a *context*.

Examples:

Suppose that Γ maps x to 0 and y to 1.

- Representation of $x\ y$ is 0 1
- Representation of $\lambda z.\ x\ y\ z$ $\lambda.$ 1 2 0

Shifting

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

$$
\begin{aligned}
\uparrow^i_c (n) &= \begin{cases} n & \text{if } n < c \\ n+i & \text{otherwise} \end{cases} \\
\uparrow^i_c (\lambda.e) &= \lambda.(\uparrow^i_{c+1} e) \\
\uparrow^i_c (e_1\, e_2) &= (\uparrow^i_c e_1)\,(\uparrow^i_c e_2)
\end{aligned}
$$

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:

$$
\begin{array}{rcl}
n\{e/m\} & = & \begin{cases} e & \text{if } n = m \\ n & \text{otherwise} \end{cases} \\
(\lambda.e_1)\{e/m\} & = & \lambda.e_1\{(\uparrow^1_0 e)/m+1\} \\
(e_1\, e_2)\{e/m\} & = & (e_1\{e/m\})\,(e_2\{e/m\})
\end{array}
$$

Substitution

Now we can define substitution:

$$
\begin{array}{rcl}
n\{e/m\} & = & \begin{cases} e & \text{if } n = m \\ n & \text{otherwise} \end{cases} \\
(\lambda.e_1)\{e/m\} & = & \lambda.e_1\{(\uparrow^1_0 e)/m+1\} \\
(e_1\, e_2)\{e/m\} & = & (e_1\{e/m\})\,(e_2\{e/m\})
\end{array}
$$

The β rule for terms in de Bruijn notation is just:

$$
\frac{}{(\lambda.e_1)\, e_2 \;\rightarrow\; \uparrow^{-1}_0 (e_1\{\uparrow^1_0 e_2/0\})}\;\beta
$$

Example

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 $\Gamma(y) = 1$.

$$\begin{array}{rl} & (\lambda.\lambda.1\, 2)\, 1 \\ \rightarrow & \uparrow_0^{-1} ((\lambda.1\, 2)\{(\uparrow_0^1 1)/0\}) \end{array}$$

Example

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

$$
\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\rightarrow & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\})
\end{array}
$$

Example

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

$$
\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\to & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{(\uparrow_0^1 2)/(0+1)\})
\end{array}
$$

Example

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

$$
\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\to & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{(\uparrow_0^1 2)/(0+1)\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{3/1\})
\end{array}
$$

Example

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

$$\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\to & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{(\uparrow_0^1 2)/(0+1)\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{3/1\}) \\
= & \uparrow_0^{-1} \lambda.(1\{3/1\})\,(2\{3/1\})
\end{array}$$

Example

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

$$
\begin{array}{ll}
 & (\lambda.\lambda.1\,2)\,1 \\
\to & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{(\uparrow_0^1 2)/(0+1)\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{3/1\}) \\
= & \uparrow_0^{-1} \lambda.(1\{3/1\})\,(2\{3/1\}) \\
= & \uparrow_0^{-1} \lambda.3\,2
\end{array}
$$

Example

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

$$
\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\to & \uparrow_0^{-1} ((\lambda.1\,2)\{(\uparrow_0^1 1)/0\}) \\
= & \uparrow_0^{-1} ((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{(\uparrow_0^1 2)/(0+1)\}) \\
= & \uparrow_0^{-1} \lambda.((1\,2)\{3/1\}) \\
= & \uparrow_0^{-1} \lambda.(1\{3/1\})\,(2\{3/1\}) \\
= & \uparrow_0^{-1} \lambda.3\,2 \\
= & \lambda.2\,1
\end{array}
$$

Example

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

$$
\begin{array}{rl}
 & (\lambda.\lambda.1\,2)\,1 \\
\rightarrow & \uparrow_0^{-1}\,((\lambda.1\,2)\{(\uparrow_0^{1}\,1)/0\}) \\
= & \uparrow_0^{-1}\,((\lambda.1\,2)\{2/0\}) \\
= & \uparrow_0^{-1}\,\lambda.((1\,2)\{(\uparrow_0^{1}\,2)/(0+1)\}) \\
= & \uparrow_0^{-1}\,\lambda.((1\,2)\{3/1\}) \\
= & \uparrow_0^{-1}\,\lambda.(1\{3/1\})\,(2\{3/1\}) \\
= & \uparrow_0^{-1}\,\lambda.3\,2 \\
= & \lambda.2\,1
\end{array}
$$

which, in standard notation (with respect to Γ), 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 λ-calculus is to work with closed expressions or *combinators*.

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

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.

$$
\begin{aligned}
\mathsf{K} &= \lambda x. \lambda y.\, x \\
\mathsf{S} &= \lambda x. \lambda y. \lambda z.\, x\, z\, (y\, z) \\
\mathsf{I} &= \lambda x.\, x
\end{aligned}
$$

Combinators

We can even define independent evaluation rules that don't depend on the λ-calculus at all.

Behold the "SKI-calculus":

$$\mathsf{K}\ e_1\ e_2 \to e_1$$
$$\mathsf{S}\ e_1\ e_2\ e_3 \to e_1\ e_3\ (e_2\ e_3)$$
$$\mathsf{I}\ e \to e$$

You would never want to program in this language—it doesn't even have variables!—but it's exactly as powerful as the λ-calculus.

Bracket Abstraction

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

$$\begin{array}{rcll} [x]\,x & = & \mathsf{I} & \\ [x]\,N & = & \mathsf{K}\,N & \text{where } x \notin \mathit{fv}(N) \\ [x]\,N_1\,N_2 & = & \mathsf{S}\,([x]\,N_1)\,([x]\,N_2) & \end{array}$$

The idea is that $([x]\,M)\,N \to M\{N/x\}$ for every term N.

Bracket Abstraction

We then define a function $(e)*$ that maps a λ-calculus expression to a combinator term:

$$
\begin{aligned}
(x)* &= x \\
(e_1\ e_2)* &= (e_1)*\ (e_2)* \\
(\lambda x.e)* &= [x]\ (e)*
\end{aligned}
$$

Example

As an example, the expression $\lambda x. \lambda y.\, x$ is translated as follows:

$$
\begin{aligned}
& (\lambda x. \lambda y.\, x)* \\
= \quad & [x]\, (\lambda y.\, x)* \\
= \quad & [x]\, ([y]\, x) \\
= \quad & [x]\, (\mathsf{K}\, x) \\
= \quad & (\mathsf{S}\, ([x]\, \mathsf{K})\, ([x]\, x)) \\
= \quad & \mathsf{S}\, (\mathsf{K}\, \mathsf{K})\, \mathsf{I}
\end{aligned}
$$

No variables in the final combinator term!

Example

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

$$\begin{array}{rl} & (\lambda x. \lambda y.\, x)\; e_1\; e_2 \\ \to & (\lambda y.\, e_1)\; e_2 \\ \to & e_1 \end{array}$$

Example

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

$$\begin{array}{ll} & (\lambda x.\lambda y.\, x)\; e_1\; e_2 \\ \rightarrow & (\lambda y.\, e_1)\; e_2 \\ \rightarrow & e_1 \end{array}$$

and

$$\begin{array}{ll} & (\mathsf{S}\;(\mathsf{K}\;\mathsf{K})\;\mathsf{I})\; e_1\; e_2 \\ \rightarrow & (\mathsf{K}\;\mathsf{K}\; e_1)\;(\mathsf{I}\; e_1)\; e_2 \\ \rightarrow & \mathsf{K}\; e_1\; e_2 \\ \rightarrow & e_1 \end{array}$$

SKI Without I

Looking back at our definitions...

$$\begin{aligned}
&\mathsf{K}\ e_1\ e_2 \rightarrow e_1 \\
&\mathsf{S}\ e_1\ e_2\ e_3 \rightarrow e_1\ e_3\ (e_2\ e_3) \\
&\mathsf{I}\ e \rightarrow e
\end{aligned}$$

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

SKI Without I

Looking back at our definitions...

$$\mathsf{K}\ e_1\ e_2 \to e_1$$
$$\mathsf{S}\ e_1\ e_2\ e_3 \to e_1\ e_3\ (e_2\ e_3)$$
$$\mathsf{I}\ e \to e$$

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

Our example becomes:

$$\mathsf{S}\ (\mathsf{K}\ \mathsf{K})\ (\mathsf{S}\ \mathsf{K}\ \mathsf{K})$$