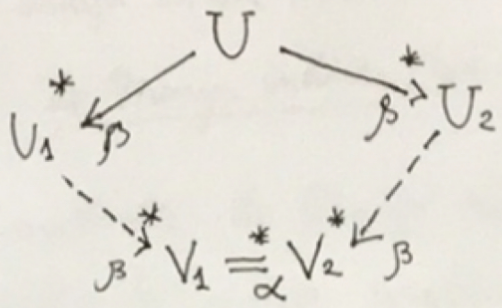
Machine-friendly λ Calculus: de Bruijn Numbers

CS 524
Lecture 15
J. Meseguer

The λ-calculus notation is user-friendly, but not machine-friendly. Furthermore, it has an additional non-trivial drawback: the λ-Calculus is deterministic, and therefore a functional formalism. However, the unique result yielded by a terminating λ-expression U is not unique just up to β-reduction or β,η-reduction, but only up to α-conversion of the results of such reduction. This can be summarized by saying [for β-reduction: the case for β,η is similar] that it has the Church-Rosser property: Given any U, and β-reduction sequences $U \xrightarrow[\beta]{*} U_1$, $U \xrightarrow[\beta]{*} U_2$ there are terms V_1, V_2 and β-reduction sequences $U_1 \xrightarrow[\beta]{*} V_1$, $U_2 \xrightarrow[\beta]{*} V_2$ such that:



For example, the result of β-reducing U may be the identity function term $V_1 = \lambda x.\, x$ along the sequence $U \xrightarrow[\beta]{*} U_1 \xrightarrow[\beta]{*} V_1$, and the equivalent identity function term $V_2 = \lambda y.\, y$ along the sequence $U \xrightarrow[\beta]{*} U_2 \xrightarrow{*} V_2$, so that $\lambda x.\, x \overset{*}{=\!=}_{\alpha} \lambda y.\, y$, where $\overset{*}{=\!=}_{\alpha}$ denotes application of $0, 1, \ldots, n$, steps of the α-equivalence rule. The practical problem is that:

> We still have left the problem of comparing equality of results in the λ-calculus, which requires α-equivalence equality reasoning.

Wouldn't it be nice if we can have a version of the λ-calculus that:

1. is machine-friendly, and, where
2. equality of results becomes syntactic identity

This is what de Bruijn in the Netherlands accomplished by the use of de Bruijn indices [or numbers].

But to better motivate de Bruijn notation we can first look at the following problem:

3 Closed

A Canonical Form for λ-Terms up to α-equivalence

Goal: Given a λ-term U, define an α-equivalent term $can(U)$ such that:

1. $U \overset{*}{=}_\alpha can(U)$
2. If $U \overset{*}{=}_\alpha V$, then $can(U) \equiv can(V)$

where $\_\equiv\_$ denotes syntactic equality

We can define $can(U)$ quite easily by taking advantage of the assumption that names can be linearly ordered.
Let us choose names/variables in the set $\{x_{[n]} \mid n \in \mathbb{N}\}$.
Then can be linearly ordered as:

$$x_{[0]} < x_{[1]} < x_{[2]} < \cdots < x_{[n]} < \cdots$$

Then the key idea is the notion of the λ-depth of a λ-subexpression. ~~For example, in~~ of the form $\lambda x.U$

For example, in $\lambda x.\ \lambda y.\ ((\lambda z.(x\ z))\ (\lambda v.((x\ y)\ v)))$

the entire term has λ-depth 0, the term

- $\lambda y.((\lambda z.(x\ z))\ (\lambda v.(x\ y)\ v))$ has λ-depth 1, and
- $\lambda z.(x\ z)$ and $\lambda v.((x\ y)\ v)$ have λ-depth 2

For simplicity, let us assume that the set Vars of names/variables in λ-expressions is <u>disjoint</u> from the ~~set~~ set of names

$Vars' = \{ x_{\{0\}}, x_{\{1\}}, \ldots, x_{\{n\}}, \ldots \}$ where the canonical form $can(U)$ for U with names in Vars will be defined, so that for U, U' terms $U \overset{*}{=}_\alpha U'$ and such that $fv(U) = fv(U') = \emptyset$ [<u>closed</u> λ-terms] we will always have: (1) $can(U) = can(U')$, (2) $U \overset{*}{=}_\alpha can(U) =_\alpha can(U')$

where (2) makes sense for the disjoint union of names $Vars \dot\cup Vars'$.

<u>We define</u>: $can(U) = can\text{-}aux(U, 0)$

$can\text{-}aux(\lambda X.\, U, n) = [X := x_{\{n\}}](can\text{-}aux(U, n+1))$

$can\text{-}aux((U\ V), n) = can\text{-}aux(U, n)\ \ can\text{-}aux(V, n)$

$can\text{-}aux(X, n) = X$ [X ranging over Vars]

<u>Therefore</u>: $can(\lambda x.\, \lambda y.((\lambda z.(x\ z))\ \lambda v.((x\ y)\ v))) =$

$\lambda x_{\{0\}} \cdot \lambda x_{\{1\}} \cdot [x := x_{\{0\}}][y := x_{\{1\}}](\lambda x_{\{2\}}[z := x_{\{0\}}](x, z))$

$\lambda x_{\{2,3\}}[v := x_{\{2,3\}}]((x\ y)\ v)) =$

$\lambda x_{\{0\}} \cdot \lambda x_{\{1\}} \cdot (\lambda x_{\{2\}}[x := x_0][y := x_{\{1\}}][z := x_{\{2\}}](x\ z)$

$\lambda x_{\{2,3\}}[x := x_0][y := x_1][v := x_{\{2,3\}}]((x\ y)\ v)) =$

$\lambda x_{\{0\}} \cdot \lambda x_{\{1\}} \cdot ((\lambda x_{\{2,3\}} \cdot (x_{\{0\}}\ x_{\{2,3\}}))\ \lambda x_{\{2,3\}}((x_{\{0\}}\ x_{\{1\}})\ x_{\{2,3\}}))$

The de Bruijn Notation will easily transform a term of the form $\text{can}(U)$, where U is a closed λ-term, by: replacing each $\lambda x_{\{n\}}.$ by a $\lambda.$, and each $x_{\{j\}}$ by a natural numbers as follows. Define for each occurrence of a variable $x_{\{j\}}$:

$\begin{cases} k = \lambda\text{-depth} = \text{number of } \lambda\text{'s that } x_{\{j\}} \text{ is under} \\ d = \lambda\text{-distance} = \text{number of } \lambda\text{'s crossed to reach binder } \lambda x_{\{j\}} \end{cases}$

The translation for terms with names in Vars' also applies to terms with free variables. It replaces:

(1) a bound $x_{\{j\}}$ by d and (2) a free $x_{\{j\}}$ by $j+k$

For example:

(1) For the closed term: $U = \lambda x_{\{0\}} . \lambda x_{\{1\}} . ((\lambda x_{\{2\}} . (x_{\{0\}}\ x_{\{2\}}))\ \lambda x_{\{2\}} . ((x_{\{0\}}\ x_{\{1\}})\ x_{\{2\}}))$

$d=2\ d=0 \qquad d=2\ d=1\ d=0$

$$dB(U) = \lambda.\ \lambda.\ ((\lambda.\ (2\ 0))\ \lambda.((2\ 1)\ 0))$$

(2) For V with free variables:

$$V = (x_{\{3\}}\ x_{\{2\}})\ \lambda x_{\{0\}} . ((x_{\{0\}}\ x_{\{2\}})\ (\lambda x_{\{1\}} . (x_{\{1\}}\ x_{\{3\}})))$$

$k=0\ k=0 \qquad k=1\ k=1 \qquad k=2\ k=2$

$d=0 \qquad d=0$

$$dB(V) = (3\ 2)\ \lambda.((0\ 3)\ (\lambda.(0\ 5)))$$

Note that no information is lost in the dB translation. We can recover the original term by replacing:

① Bound variable $n < k$ at depth k by: $x_{\{k-(n+1)\}}$

② Free variable $n \geq k$ at λ-depth $k \geq 0$ by: $x_{\{n-k\}}$

Example: $\underset{K=0}{(3}\ \underset{k=0}{2)}\ \lambda.((\underset{K=1}{0}\ \underset{K=1}{3})(\lambda.(\underset{K=2}{0}\ \underset{K=2}{5})))$

Becomes: $(x_{\{3\}}\ x_{\{2\}})\ \lambda x_{\{0\}}((x_{\{0\}}\ x_{\{2\}})(\lambda x_{\{1\}}.(x_{\{1\}}\ x_{\{3\}})))$

β-Reduction in de Bruijn Notation has the form:

$\beta : (\lambda.U)\ V \longrightarrow [V]\ U$, where "intuitively" $[V]$ is $[x_{\{0\}} := V]$

For example: $(\lambda.\lambda.((3\ 1)(4\ 0)))(\lambda.(0\ 0)) \underset{\beta}{\rightarrow} [\lambda.(0\ 0)]\ \lambda.((3\ 1)(4\ 0))$

Note that, since we have removed the outer $\lambda.$, the term $\lambda.((3\ 1)(4\ 0))$ is not a proper de Bruijn term: we need to adjust the indices for the free variables (3 and 4) down by 1, so it's really the term in dB notation: $\lambda.((2\ 1)(3\ 0))$

which, according to ② above, has free variables $x_{\{0\}}, x_{\{1\}}, x_{\{2\}}$ in the standard notation. But the crucial free variable is $1 = x_{\{0\}}$, since this is the one that was bound by the outer lambda and needs to be substituted: $[\lambda.(0\ 0)]$ really means $[x_{\{0\}} := \lambda.(0\ 0)]$ in "mixed" notation.

Therefore, we would need to replace the free variable 1 in term $\lambda.((2\ 1)(3\ 0))$ by $\lambda.(0\ 0)$, yielding the term: $\lambda.((2\ \lambda.(0\ 0))(3\ 0))$.

So we should get:

$$[\lambda.(0\ 0)]\,(\lambda.(3\ 1)(4\ 0)) = \lambda.((2\ \lambda.(0\ 0))(3\ 0))$$

How can we do that? By applying the equations in the following equational definition of dB substitution:
[see Appendix lecture15.A1.pdf for the Maude module]

```
op [_] : LambdaDB -> Sub [frozen] .     *** substitution
op [^] : -> Sub .                       *** shift
op [^_] : Sub -> Sub [frozen] .         *** lift

vars N M : LambdaDB . var n : Nat . var SU : Sub .

eq [1] : [M] 0 = M                      *** substitution on free var "x_203"
eq [2] : [M] s(n) = n                   *** substitution on var ≠ "x_203"
eq [3] : [^] n = s(n)                   *** shift to adjust for enclosing λ's.
eq [4] : [^ SU] 0 = 0                   *** substitution on bound variable
```

eq [5] : $[\uparrow SU]\ 1(n) = [\uparrow]\ (SU\ n)$ *** shif (later)

*** and lift variable 1(n)

*** under a λ

eq [6] : $SU\ (M\ N) = (SU\ M)\ (SU\ N)$.

blank space here

eq [7] : $SU(\lambda.\ M) = \lambda.\ ([\uparrow SU]\ M)$

Let us apply these equations to our example problem:

$[\lambda.(0\ 0)]\ (\lambda.((3\ 1)\ (4\ 0))) =_{[7]} \lambda.\ [\uparrow [\lambda.(0\ 0)]((3\ 1)\ (4\ 0)) =_{[6]}$

$= \lambda.\ ([\uparrow [\lambda.(0\ 0)]\ (3\ 1)\ [\uparrow [\lambda.(0\ 0)]\ (4\ 0)) =_{[6]}$

$= \lambda.\ ([\uparrow [\lambda.(0\ 0)]\ 3\ [\uparrow [\lambda.(0\ 0)]\ 1)\ ([\uparrow [\lambda.(0\ 0)]\ 4\ [\uparrow [\lambda.(0\ 0)]\ 0) =$ [4,5]

$= \lambda.\ ([\uparrow]\ [\lambda.(0\ 0)]\ 2\ [\uparrow]\ [\lambda.(0\ 0)]\ 0)\ ([\uparrow]\ [\lambda.(0\ 0)]\ 3\ \ldots\ 0) =$ [2,1]

$= \lambda.\ ([\uparrow]\ 1\ [\uparrow]\ \lambda.(0\ 0))\ ([\uparrow]\ 2\ 0) =_{[3,7]}$

$= \lambda.\ ((2\ \lambda.([\uparrow [\uparrow]\ 0\ [\uparrow [\uparrow]\ 0))\ (3,0)) =$ [4]

$= \lambda.\ ((2\ \ldots\ \lambda.(0\ 0))\ (3\ 0))$

which is the result we expected, corresponding to the term:

$\lambda x_{\{0\}}.\ ((x_{\{1\}}\ \lambda x_{\{1\}}.(x_{\{1\}}\ x_{\{1\}}))\ (x_{\{2\}}\ x_{\{0\}}))$

with free variables: $x_{\{1\}}$ and $x_{\{2\}}$.

Compiling λ-expressions into de Bruijn Notation

Since de Bruijn notation is:

1. machine friendly, and closed
2. has unique representation for terms, and
   therefore syntactic uniqueness of results of β-reduction,

it is an attractive alternative to combinators for implementing the λ-calculus, because λ-expressions in de Bruijn notation are still human-readable.

Appendix in lecture15-A1.pdf defines in Maude such a compilation function:

λ.2DB : Lambda{N} → ~~Lambda~~ DB

independently of the choice of a totally-ordered set N of names, i.e., in a parameterized theory extending the parameterized theory of λ-expressions with names, by ~~adding~~ importing also ~~a new sort for~~ the de Bruijn λ-terms.

That is, we need not require that the names are totally ordered: only that it is an infinite countable set.