De Bruijn’s syntax and reductional equivalence of \(\lambda\)-terms

Fairouz Kamareddine (Heriot-Watt University)
Roel Bloo and Rob Nederpelt (Eindhoven, NL)

5 September 2001
Item Notation/Lambda Calculus à la de Bruijn

- $\mathcal{I}$ translates to item notation:
  \[
  \mathcal{I}(x) = x, \quad \mathcal{I}(\lambda x. B) = [x]\mathcal{I}(B), \quad \mathcal{I}(AB) = (\mathcal{I}(B))\mathcal{I}(A)
  \]

- $(\lambda x.\lambda y.xy)z$ translates to $(z)[x][y](y)x$.

- The items are $(z)$, $[x]$, $[y]$ and $(y)$. The last $x$ is the heart of the term.

- The applicator wagon $(z)$ and abstractor wagon $[x]$ occur NEXT to each other.

- The $\beta$ rule $(\lambda x. A)B \rightarrow_\beta A[x := B]$ becomes in item notation:
  \[
  (B)[x]A \rightarrow_\beta [x := B]A
  \]
Redexes in Item Notation

Classical Notation

\[
\begin{align*}
((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a & \rightarrow^\beta ((\lambda_y.\lambda_z.zd)c)a \\
((\lambda_y.\lambda_z.zd)c)a & \rightarrow^\beta (\lambda_z.zd)a \\
(\lambda_z.zd)a & \rightarrow^\beta ad
\end{align*}
\]

Item Notation

\[
\begin{align*}
(a)(b)[x](c)[y][z](d)z & \rightarrow^\beta (a)(c)[y][z](d)z \\
(a)(c)[y][z](d)z & \rightarrow^\beta (a)[z](d)z \\
(a)[z](d)z & \rightarrow^\beta (d)a
\end{align*}
\]
Segments, Partners, Bachelors

- The “bracketing structure” of \((\lambda_x.(\lambda_y.\lambda_z. \ldots c)b)a\), is \(\{1 \{2 \{3 \}2 \}1 \}3\), where \(\{i\} \) and \(\{i\}_i\) match.

- The bracketing structure of \((a)(b)[x](c)[y][z](d)\) is simpler: \(\{\}\{\}\{\}\).  

- \((a)\) and \([z]\) are partners. \((b)\) and \([x]\) are partners. \((c)\) and \([y]\) are partners.

- \((d)\) is bachelor.

- A segment \(\vec{s}\) is well balanced when it contains only partnered main items. \((a)(b)[x](c)[y][z]\) is well balanced.

- A segment is bachelor when it contains only bachelor main items.
More on Segments, Partners, and Bachelors

- The \textit{main} items are those at top level.
  In \((\{y\}(y)y)x\) the main items are: \((\{y\}(y)y)\) and \([x]\).
  \([y]\) and \((y)\) are \textit{not} main items.

- Each main bachelor \([\times]\) precedes each main bachelor \((\times)\).
  For example, look at: \([u](a)(b)[x](c)[y][z](d)u\).

- Removing all main bachelor items yields a well balanced segment.
  For example from \([u](a)(b)[x](c)[y][z](d)\) we get: \((a)(b)[x](c)[y][z]\).

- Removing all main partnered items yields a bachelor segment \([v_1]\ldots[v_n](a_1)\ldots(a_m)\).
  For example from \([u](a)(b)[x](c)[y][z](d)\) we get: \([u](d)\).

- If \([v]\) and \((b)\) are partnered in \(\overline{s_1} (b) \overline{s_2} [v] \overline{s_3}\), then \(\overline{s_2}\) must be well balanced.
Even More on Segments, Partners, and Bachelors

Each non-empty segment $\overline{s}$ has a unique \textit{partitioning} into sub-segments $\overline{s} = \overline{s_0} \overline{s_1} \cdots \overline{s_n}$ such that $n \geq 0$,

• $\overline{s_i}$ is not empty for $i \geq 1$,

• $\overline{s_i}$ is well balanced if $i$ is even and is bachelor if $i$ is odd.

• if $\overline{s_i} = [x_1] \cdots [x_m]$ and $\overline{s_j} = (a_1) \cdots (a_p)$ then $\overline{s_i}$ precedes $\overline{s_j}$

• Example: $\overline{s} \equiv [x][y](a)[z][x'][b](c)(d)[y'][z'][e]$ is partitioned as:

$\overline{s} \equiv \underbrace{\overline{s_0}}_{\emptyset} \underbrace{\overline{s_1}}_{[x][y]} \underbrace{\overline{s_2}}_{(a)} \underbrace{\overline{s_3}}_{[z]} \underbrace{\overline{s_4}}_{[x'][b]} \underbrace{\overline{s_5}}_{(c)(d)[y'][z']}$
More on Item Notation

- Above discussion and further details of item notation can be found in [Kamareddine and Nederpelt, 1995, 1996].

- Item notation helped greatly in the study of a one-sorted style of explicit substitutions, the λs-style which is related to λσ, but has certain simplifications [Kamareddine and Ríos, 1995, 1997; Kamareddine and Ríos, 2000].

- For explicit substitution in item notation see [Kamareddine and Nederpelt, 1993]
Canonical Forms

- Nice canonical forms look like:

<table>
<thead>
<tr>
<th>bachelor [ ]s</th>
<th>( )[]-pairs, $A_i$ in CF</th>
<th>bachelor ( )s, $B_i$ in CF</th>
<th>end var</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x_1] \ldots [x_n]$</td>
<td>$(A_1)[y_1] \ldots (A_m)[y_m]$</td>
<td>$(B_1) \ldots (B_p)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

- Classical:

$$\lambda x_1 \ldots \lambda x_n.(\lambda y_1.(\lambda y_2.\ldots(\lambda y_m. x B_p \ldots B_1)A_m \ldots)A_2)A_1$$

- For example, a canonical form of:

$$[x][y](a)[z][x'](b)(c)(d)[y'][z'](e)x$$

is

$$[x][y][x'](a)[z](d)[y'](c)[z'](b)(e)x$$
Some Helpful Rules for reaching canonical forms

<table>
<thead>
<tr>
<th>Name</th>
<th>In Classical Notation</th>
<th>In Item Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(θ)</td>
<td>(((\lambda_x \cdot N) P) Q)</td>
<td>((Q)(P)[x]N)</td>
</tr>
<tr>
<td></td>
<td>((\lambda_x \cdot N Q) P)</td>
<td>((P)<a href="Q">x</a>N)</td>
</tr>
<tr>
<td>(γ)</td>
<td>((\lambda_x \cdot \lambda_y \cdot N) P)</td>
<td>((P)[x][y]N)</td>
</tr>
<tr>
<td></td>
<td>((\lambda_y \cdot (\lambda_x \cdot N) P)</td>
<td>(<a href="P">y</a>[x]N)</td>
</tr>
<tr>
<td>(γC)</td>
<td>(((\lambda_x \cdot \lambda_y \cdot N) P) Q)</td>
<td>((Q)(P)[x][y]N)</td>
</tr>
<tr>
<td></td>
<td>((\lambda_y \cdot (\lambda_x \cdot N) P) Q)</td>
<td>((Q)<a href="P">y</a>[x]N)</td>
</tr>
<tr>
<td>(g)</td>
<td>(((\lambda_x \cdot \lambda_y \cdot N) P) Q)</td>
<td>((Q)(P)[x][y]N)</td>
</tr>
<tr>
<td></td>
<td>((\lambda_x \cdot N[y := Q]) P)</td>
<td>((P)[x][y := Q]N)</td>
</tr>
</tbody>
</table>
A Few Uses of Generalised Reduction and Term Reshuffling


- Term reshuffling is used in [Kfoury et al., 1994; Kfoury and Wells, 1994] in analyzing typability problems.

- [Nederpelt, 1973; de Groote, 1993; Kfoury and Wells, 1995] use generalised reduction and/or term reshuffling in relating SN to WN.

- [Ariola et al., 1995] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.

- [Kamareddine and Nederpelt, 1995; Kamareddine et al., 1999, 1998; Bloo et al., 1996] shows that they could reduce space/time needs.

- [Kamareddine, 2000] shows various strong properties of generalised reduction.
## Obtaining Canonical Forms

<table>
<thead>
<tr>
<th>( \theta )-nf:</th>
<th>([\ ])-pairs mixed with bach. ([\ ])s</th>
<th>bach. ((\ ))s</th>
<th>end var (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_1)[x][y]<a href="A_2">z</a>[p]\ldots</td>
<td>(B_1)(B_2)\ldots</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \gamma )-nf:</th>
<th>([\ ])-pairs mixed with bach. ((\ ))s</th>
<th>end var (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>([x_1][x_2]\ldots )</td>
<td>((B_1)(A_1)<a href="B_2">x</a>\ldots )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \theta - \gamma )-nf:</th>
<th>([\ ])-pairs</th>
<th>bach. ((\ ))s</th>
<th>end var (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>([x_1][x_2]\ldots )</td>
<td>((A_1)<a href="A_2">y_1</a>[y_2]\ldots (A_m)[y_m])</td>
<td>((B_1)(B_2)\ldots )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \gamma - \theta )-nf:</th>
<th>([\ ])-pairs</th>
<th>bach. ((\ ))s</th>
<th>end var (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>([x_1][x_2]\ldots )</td>
<td>((A_1)<a href="A_2">y_1</a>[y_2]\ldots (A_m)[y_m])</td>
<td>((B_1)(B_2)\ldots )</td>
<td></td>
</tr>
</tbody>
</table>
**Example**

For $M \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)x$:

<table>
<thead>
<tr>
<th>$\theta(M)$:</th>
<th>bach. []s</th>
<th>()[]-pairs mixed with bach. []s</th>
<th>bach. ()s</th>
<th>end var</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x][y]$</td>
<td></td>
<td>$(a)[z]<a href="d">x'</a><a href="c">y'</a>[z']$</td>
<td>$(b)(e)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\gamma(M)$:</th>
<th>bach. []s</th>
<th>()[]-pairs mixed with bach. ()s</th>
<th>bach. ()s</th>
<th>end var</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x][y][x']$</td>
<td></td>
<td>$(a)<a href="b">z</a>(c)<a href="d">z'</a>[y']$</td>
<td>$(e)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\theta(\gamma(M))$:</th>
<th>bach. []s</th>
<th>()[]-pairs</th>
<th>bach. ()s</th>
<th>end var</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x][y][x']$</td>
<td></td>
<td>$(a)<a href="c">z</a><a href="d">z'</a>[y']$</td>
<td>$(b)(e)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\gamma(\theta(M))$:</th>
<th>bach. []s</th>
<th>()[]-pairs</th>
<th>bach. ()s</th>
<th>end var</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x][y][x']$</td>
<td></td>
<td>$(a)<a href="d">z</a><a href="c">y'</a>[z']$</td>
<td>$(b)(e)$</td>
<td>$x$</td>
</tr>
</tbody>
</table>
Classes of terms modulo reductional behaviour

- $\rightarrow_{\theta}$ and $\rightarrow_{\gamma}$ are SN and CR. Hence $\theta$-nf and $\gamma$-nf are unique.

- Both $\theta(\gamma(A))$ and $\gamma(\theta(A))$ are in canonical form.

- $\theta(\gamma(A)) =_{p} \gamma(\theta(A))$ where $\rightarrow_{p}$ is the rule

$$(A_1)[y_1](A_2)[y_2]B \rightarrow_{p} (A_2)[y_2](A_1)[y_1]B \quad \text{if } y_1 \notin \text{FV}(A_2)$$

- We define: $[A]$ to be $\{B \mid \theta(\gamma(A)) =_{p} \theta(\gamma(B))\}$.

- When $B \in [A]$, we write that $B \simeq_{\text{equi}} A$.

- $\rightarrow_{\theta}, \rightarrow_{\gamma}, =_{\gamma}, =_{\theta}, =_{p} \subseteq \simeq_{\text{equi}} \subseteq =_{\beta}$ (strict inclusions).

- Define $\text{CCF}(A)$ as $\{A' \text{ in canonical form} \mid A' =_{p} \theta(\gamma(A))\}$. 

PPDP’01, 5 September 2001
Reduction based on classes

- One-step class-reduction $\sim_{\beta}$ is the least compatible relation such that:
  \[ A \sim_{\beta} B \quad \text{iff} \quad \exists A' \in [A], \exists B' \in [B]. A' \rightarrow_{\beta} B' \]

- $\sim_{\beta}$ really acts as reduction on classes:

- If $A \sim_{\beta} B$ then forall $A' \approx_{\text{equi}} A$, forall $B' \approx_{\text{equi}} B$, we have $A' \sim_{\beta} B'$. 
Properties of reduction modulo classes

• $\sim_{\beta}$ generalises $\rightarrow_g$ and $\rightarrow_{\beta}$: $\rightarrow_{\beta} \subseteq \rightarrow_g \subseteq \sim_{\beta} \subseteq =_{\beta}$.

• $\approx_{\beta}$ and $=_{\beta}$ are equivalent: $A \approx_{\beta} B$ iff $A =_{\beta} B$.

• $\mathrel{\sim_{\beta}}$ is Church Rosser:
  If $A \mathrel{\sim_{\beta}} B$ and $A \mathrel{\sim_{\beta}} C$, then for some $D$: $B \mathrel{\sim_{\beta}} D$ and $C \mathrel{\sim_{\beta}} D$.

• Classes preserve $SN_{\rightarrow_{\beta}}$: If $A \in SN_{\rightarrow_{\beta}}$ and $A' \in [A]$ then $A' \in SN_{\rightarrow_{\beta}}$.

• Classes preserve $SN_{\sim_{\beta}}$: If $A \in SN_{\sim_{\beta}}$ and $A' \in [A]$ then $A' \in SN_{\sim_{\beta}}$.

• $SN_{\rightarrow_{\beta}}$ and $SN_{\sim_{\beta}}$ are equivalent: $A \in SN_{\sim_{\beta}}$ iff $A \in SN_{\rightarrow_{\beta}}$. 

PPDP’01, 5 September 2001
Using Item Notation in Type Systems

- Now, all items are written inside () instead of using () and [].
- \( (\lambda_x.x)y \) is written as: \((y\delta)(\lambda_x)x\) instead of \((y)[x]x\).
- \( \Pi_{z:*.}(\lambda_{x:z}.x)y \) is written as: \((*\Pi_z)(y\delta)(z\lambda_x)x\).
The Barendregt Cube in item notation and class reduction

- The formulation is the same except that terms are written in item notation:

\[ \mathcal{T} = \star \mid \Box \mid V \mid (\mathcal{T}\delta)\mathcal{T} \mid (\mathcal{T}\lambda_V)\mathcal{T} \mid (\mathcal{T}\Pi_V)\mathcal{T}. \]

- The typing rules don’t change although we do class reduction \( \rightsquigarrow_\beta \) instead of normal \( \beta \)-reduction \( \rightarrow_\beta \).

- The typing rules don’t change because \( =_\beta \) is the same as \( \approx_\beta \).
Figure 1: The Barendregt Cube

\[
\begin{align*}
\lambda_2 & \quad \lambda P_2 \\
\lambda\omega & \quad \lambda P\omega
\end{align*}
\]
Subject Reduction fails

- Most properties including SN hold for all systems of the cube extended with class reduction. However, SR only holds in $\lambda \rightarrow (\ast, \ast)$ and $\lambda \omega (\blacksquare, \blacksquare)$.

- SR fails in $\lambda P (\ast, \blacksquare)$ (and hence in $\lambda P 2, \lambda P \omega$ and $\lambda C$). Example in paper.

- SR also fails in $\lambda 2 (\blacksquare, \ast)$ (and hence in $\lambda P 2, \lambda \omega$ and $\lambda C'$):
Why does Subject Reduction fails

- \((y' \delta)(\beta \delta)(\lambda_\alpha)(\alpha \lambda_y)(y \delta)(\alpha \lambda_x)x \sim_\beta (\beta \delta)(\lambda_\alpha)(y' \delta)(\alpha \lambda_x)x\).

- \((\lambda_\alpha:*.(\lambda_{y:}\alpha.(\lambda_{x:}\alpha.x)y)y')\beta' \sim_\beta (\lambda_\alpha:*.(\lambda_{x:}\alpha.x)y')\beta\)

- \(\beta : *, y' : \beta \vdash_2 (\lambda_\alpha:*.(\lambda_{y:}\alpha.(\lambda_{x:}\alpha.x)y)y')\beta' : \beta\)

- Yet, \(\beta : *, y' : \beta \vdash_2 (\lambda_\alpha:*.(\lambda_{x:}\alpha.x)y')\beta : \tau\) for any \(\tau\).

- the information that \(y' : \beta\) has replaced \(y : \alpha\) is lost in \((\lambda_\alpha:*.(\lambda_{x:}\alpha.x)y')\beta\).

- But we need \(y' : \alpha\) to be able to type the subterm \((\lambda_{x:}\alpha.x)y'\) of \((\lambda_\alpha:*.(\lambda_{x:}\alpha.x)y')\beta\) and hence to type \(\beta : *, y' : \beta \vdash_2 (\lambda_\alpha:*.(\lambda_{x:}\alpha.x)y')\beta : \beta\).
Solution to Subject Reduction: Use “let expressions/definitions”

- Definitions/let expressions are of the form: \( \text{let } x : A = B \) and are added to contexts exactly like the declarations \( y : C \).

- (def rule) \[
\frac{\Gamma, \text{let } x : A = B \vdash^c C : D}{\Gamma \vdash^c (\lambda x: A. C) B : D[x := A]}
\]

- we define \( \Gamma \vdash^c \cdot =_{\text{def}} \cdot \) to be the equivalence relation generated by:
  - if \( A =_{\beta} B \) then \( \Gamma \vdash^c A =_{\text{def}} B \)
  - if \( \text{let } x : M = N \) is in \( \Gamma \) and if \( B \) arises from \( A \) by substituting one particular occurrence of \( x \) in \( A \) by \( N \), then \( \Gamma \vdash^c A =_{\text{def}} B \).
The (simplified) Cube with definitions and class reduction

(axiom) (app) (abs) and (form) are unchanged.

(start)
\[ \frac{\Gamma \vdash^c A : s}{\Gamma, x : A \vdash^c x : A} \quad \frac{\Gamma \vdash^c A : s \quad \Gamma \vdash^c B : A}{\Gamma, \text{let } x : A = B \vdash^c x : A} \]

(weak)
\[ \frac{\Gamma \vdash^c D : E \quad \Gamma \vdash^c A : s}{\Gamma, x : A \vdash^c D : E} \quad \frac{\Gamma \vdash^c A : s \quad \Gamma \vdash^c B : A \quad \Gamma \vdash^c D : E}{\Gamma, \text{let } x : A = B \vdash^c D : E} \]

(conv)
\[ \frac{\Gamma \vdash^c A : B}{\Gamma \vdash^c B' : S} \quad \frac{\Gamma \vdash^c B =_{\text{def}} B'}{\Gamma \vdash^c A : B'} \]

(def)
\[ \frac{\Gamma, \text{let } x : A = B \vdash^c C : D}{\Gamma \vdash^c (\lambda x : A . C)B : D[x := A]} \]
Table 1: Definitions solve subject reduction

1. \( \beta : *, y' : \beta, \ \text{let} \ \alpha : * = \beta \quad \vdash^c y' : \beta \)

2. \( \beta : *, y' : \beta, \ \text{let} \ \alpha : * = \beta \quad \vdash^c \alpha =_{\text{def}} \beta \)

3. \( \beta : *, y' : \beta, \ \text{let} \ \alpha : * = \beta \quad \vdash^c y' : \alpha \quad \text{(from 1 and 2)} \)

4. \( \beta : *, y' : \beta, \ \text{let} \ \alpha : * = \beta, \ \text{let} \ x : \alpha = y' \quad \vdash^c x : \alpha \)

5. \( \beta : *, y' : \beta, \ \text{let} \ \alpha : * = \beta \quad \vdash^c (\lambda_{x:\alpha}.x)y' : \alpha[x := y'] = \alpha \)

\[
\beta : *, y' : \beta \quad \vdash^c (\lambda_{\alpha:*}(\lambda_{x:\alpha}.x)y')\beta : \alpha[\alpha := \beta] = \beta
\]
Properties of the Cube with definitions and class Reduction

- $\vdash^c$ is a generalisation of $\vdash$: If $\Gamma \vdash A : B$ then $\Gamma \vdash^c A : B$.

- Equivalent terms have same types:
  If $\Gamma \vdash^c A : B$ and $A' \in [A]$, $B' \in [B]$ then $\Gamma \vdash^c A' : B'$.

- Subject Reduction for $\vdash^c$ and $\leadsto^\beta$:
  If $\Gamma \vdash^c A : B$ and $A \leadsto^\beta A'$ then $\Gamma \vdash^c A' : B$.

- Unicity of Types for $\vdash^c$:
  - If $\Gamma \vdash^c A : B$ and $\Gamma \vdash^c A : B'$ then $\Gamma \vdash^c B =_{\text{def}} B'$
  - If $\Gamma \vdash^c A : B$ and $\Gamma \vdash^c A' : B'$ and $\Gamma \vdash^c A =^{\beta} A'$ then $\Gamma \vdash^c B =_{\text{def}} B'$.

- Strong Normalisation of $\leadsto^\beta$:
  In the Cube, every legal term is strongly normalising with respect to $\leadsto^\beta$. 
Bibliography


PPDP’01, 5 September 2001


