\textit{\lambda}-calculus à la Automath

Fairouz Kamareddine (Heriot-Watt University)

12 April 2002

35 Years of Automath, 12 April 2002
Item Notation/Lambda Calculus à la de Bruijn

- \( I \) translates to item notation:
  \[
  I(x) = x, \quad I(\lambda x.B) = [x]I(B), \quad I(AB) = \langle I(B) \rangle I(A)
  \]

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

- The wagons are \( \langle z \rangle \), \( [x] \), \( [y] \) and \( \langle y \rangle \). The last \( x \) is the heart of the term.

- The applicator wagon \( \langle z \rangle \) 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:
  \[
  \langle B \rangle[x]A \rightarrow_\beta [x := B]A
  \]
Redexes in Item Notation

Classical Notation

\[
\frac{((\lambda_x.(\lambda_y.\lambda_z. z \, d) \, c) \, b) \, a}{\downarrow \beta}
\]

\[
\frac{((\lambda_y.\lambda_z. z \, d) \, c) \, a}{\downarrow \beta}
\]

\[
\frac{(\lambda_z. z \, d) \, a}{\downarrow \beta}
\]

\[
ad
\]

Item Notation

\[
\langle a \rangle \langle b \rangle [x] \langle c \rangle [y] [z] \langle d \rangle z
\]

\[
\downarrow \beta
\]

\[
\langle a \rangle \langle c \rangle [y] [z] \langle d \rangle z
\]

\[
\downarrow \beta
\]

\[
\langle a \rangle [z] \langle d \rangle z
\]

\[
\downarrow \beta
\]

\[
\langle d \rangle a
\]
Segments, Partners, Bachelors

- The “bracketing structure” of \(((\lambda x.(\lambda y.\lambda z. - - )c)b)a\), is ‘\{1 \{2 \{3 \}2 \}1 \}3\}', where ‘\{i\}' and ‘\}{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 \(\bar{s}\) 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 *main* items are those at top level.
  In \((v)(y)y[x]x\) the main items are: \((y)(y)y\) and \([x]\).
  \([y]\) and \((y)\) are *not* main items.

- Each main bachelor \([\phantom{\text{main}}]\) precedes each main bachelor \((\phantom{\text{main}})\).
  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 $\overline{s} = \overline{s_0}s_1 \cdots s_n$ 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:

\[
\overline{s} \equiv \begin{array}{c}
\overline{s_0} \\
\overline{s_1} \\
\overline{s_2} \\
\overline{s_3} \\
\overline{s_4} \\
\overline{s_5}
\end{array}
\begin{array}{c}
\emptyset \\
x[y] \\
(a)[z] \\
x'[b] \\
(c)(d) \\
[y'][z'](e)
\end{array}
\]

35 Years of Automath, 12 April 2002
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 $\lambda s$-style which is related to $\lambda \sigma$, 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:

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

- 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>
</table>
| $(\theta)$ | $((\lambda_x . N) P) Q$  
$\downarrow$  
$(\lambda_x . N Q) P$ | $(Q)(P)[x]N$  
$\downarrow$  
$(P)[x](Q)N$ |
| $(\gamma)$ | $(\lambda_x . \lambda_y . N) P$  
$\downarrow$  
$\lambda_y . (\lambda_x . N) P$ | $(P)[x][y]N$  
$\downarrow$  
$[y](P)[x]N$ |
| $(\gamma C)$ | $((\lambda_x . \lambda_y . N) P) Q$  
$\downarrow$  
$(\lambda_y . (\lambda_x . N) P) Q$ | $(Q)(P)[x][y]N$  
$\downarrow$  
$(Q)[y](P)[x]N$ |
| $(g)$ | $((\lambda_x . \lambda_y . N) P) Q$  
$\downarrow$  
$(\lambda_x . N[y := Q]) P$ | $(Q)(P)[x][y]N$  
$\downarrow$  
$(P)[x][y := Q]N$ |
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., 2001, 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 ( (A_1)[x][y]<a href="A_2">z</a>[p] \ldots )</th>
<th>bach. ()s ( (B_1)(B_2) \ldots )</th>
<th>( \forall x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma )-nf:</td>
<td>bach. []s ( [x_1][x_2] \ldots ) ()[] pairs mixed with bach. ()s ( (B_1)(A_1)<a href="B_2">x</a> \ldots )</td>
<td>( \forall x )</td>
<td>( \forall x )</td>
</tr>
<tr>
<td>( \theta )-( \gamma )-nf:</td>
<td>bach. []s ( [x_1][x_2] \ldots ) ()[] pairs ( (A_1)<a href="A_2">y_1</a>[y_2] \ldots (A_m)[y_m] )</td>
<td>bach. ()s ( (B_1)(B_2) \ldots )</td>
<td>( \forall x )</td>
</tr>
<tr>
<td>( \gamma )-( \theta )-nf:</td>
<td>bach. []s ( [x_1][x_2] \ldots ) ()[] pairs ( (A_1)<a href="A_2">y_1</a>[y_2] \ldots (A_m)[y_m] )</td>
<td>bach. ()s ( (B_1)(B_2) \ldots )</td>
<td>( \forall x )</td>
</tr>
</tbody>
</table>
For $M \equiv [x][y](a)[z][x'][b](c)(d)[y'][z'][e]x$:

| $\theta(M)$: | bach. []s $[x][y]$ | ()[]-pairs mixed with bach. []s $(a)[z][x'][d][y'][c][z']$ | bach. ()s $(b)(e)$ | end var $x$ |
| $\gamma(M)$: | bach. []s $[x][y][x']$ | ()[]-pairs mixed with bach. ()s $(a)[z](b)(c)[z'][d][y']$ | bach. ()s $(e)$ | end var $x$ |
| $\theta(\gamma(M))$: | bach. []s $[x][y][x']$ | ()[]-pairs $(a)[z](c)[z'][d][y']$ | bach. ()s $(b)(e)$ | end var $x$ |
| $\gamma(\theta(M))$: | bach. []s $[x][y][x']$ | ()[]-pairs $(a)[z](d)[y'][c][z']$ | bach. ()s $(b)(e)$ | end var $x$ |
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 $\rightarrow_\rho$ is the rule

$$\begin{align*}
(A_1)[y_1](A_2)[y_2]B &\rightarrow_\rho (A_2)[y_2](A_1)[y_1]B & \text{if } y_1 \notin \text{FV}(A_2)
\end{align*}$$

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

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

• $\rightarrow_\theta$, $\rightarrow_\gamma$, $=_\gamma$, $=_\theta$, $=_p \subset \approx_{\text{equi}} \subset =_\beta$ (strict inclusions).

• Define $\text{CCF}(A)$ as $\{A' \text{ in canonical form} \mid A' =_p \theta(\gamma(A))\}$. 
Reduction based on classes [Kamareddine et al., 2001]

- One-step class-reduction $\sim_\beta$ is the least compatible relation such that:
  
  $A \sim_\beta B \iff \exists A' \in [A]. \exists B' \in [B]. A' \to_\beta B'$

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

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

• \( \leadsto_{\beta} \) generalises \( \rightarrow_{g} \) and \( \rightarrow_{\beta} \): \( \rightarrow_{\beta} \subseteq \rightarrow_{g} \subseteq \leadsto_{\beta} \subseteq =_{\beta} \).

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

• \( \leadsto_{\beta} \) is Church Rosser:
  If \( A \leadsto_{\beta} B \) and \( A \leadsto_{\beta} C \), then for some \( D \): \( B \leadsto_{\beta} D \) and \( C \leadsto_{\beta} D \).

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

• Classes preserve \( SN_{\leadsto_{\beta}} \): If \( A \in SN_{\leadsto_{\beta}} \) and \( A' \in [A] \) then \( A' \in SN_{\leadsto_{\beta}} \).

• \( SN_{\rightarrow_{\beta}} \) and \( SN_{\leadsto_{\beta}} \) are equivalent: \( A \in SN_{\leadsto_{\beta}} \) iff \( A \in SN_{\rightarrow_{\beta}} \).
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 \((y)[x]x\).

• \(\Pi z::\star.(\lambda x::z.x)y\) is written as: \((\ast\Pi z)(y\delta)(z\lambda x)x\).
The Barendregt Cube in item notation and class reduction

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

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

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

- The typing rules don’t change because \( =_\beta \) is the same as \( \cong_\beta \).
Figure 1: The Barendregt Cube
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 $\lambda \omega (\boxdot, \boxdot)$.

- SR fails in $\lambda P (\ast, \boxdot)$ (and hence in $\lambda P2, \lambda P\omega$ and $\lambda C$). Example in paper.

- SR also fails in $\lambda 2 (\boxdot, \ast)$ (and hence in $\lambda P2, \lambda \omega$ and $\lambda C'$):
Why does Subject Reduction fails

- \((y'\delta)(\beta\delta)(\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x \leadsto_{\beta}(\beta\delta)(\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x)\).

- \((\lambda_\alpha:*\lambda_y:\alpha.(\lambda_x:\alpha.x)y)\beta y' \leadsto_{\beta}(\lambda_\alpha:*.(\lambda_x:\alpha.x)y')\beta\)

- \(\beta : *, y' : \beta \vdash_2 (\lambda_\alpha:*\lambda_y:\alpha.(\lambda_x:\alpha.x)y)\beta y' : \beta\)

- Yet, \(\beta : *, y' : \beta \not\vdash_2 (\lambda_\alpha:*.(\lambda_x:\alpha.x)y')\beta : \tau\) 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 \(\beta : *, y' : \beta \vdash (\lambda_\alpha:*.(\lambda_x:\alpha.x)y')\beta : \beta\).
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 \( A \equiv^\beta B \) 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.

\[
\begin{align*}
\text{(start)} & \quad \frac{\Gamma \vdash^c A : s}{\Gamma, x : A \vdash^c x : A} & \frac{\Gamma \vdash^c A : s}{\Gamma, \text{let } x : A = B \vdash^c x : A} & x \text{ fresh} \\
\text{(weak)} & \quad \frac{\Gamma \vdash^c D : E \quad \Gamma \vdash^c A : s}{\Gamma, x : A \vdash^c D : E} & \frac{\Gamma \vdash^c A : s \quad \Gamma \vdash^c B : A \quad \Gamma \vdash^c D : E}{\Gamma, \text{let } x : A = B \vdash^c D : E} & x \text{ fresh} \\
\text{(conv)} & \quad \frac{\Gamma \vdash^c A : B}{\Gamma \vdash^c A : B'} & \frac{\Gamma \vdash^c B' : S}{\Gamma \vdash^c B =_{\text{def}} B'} & \\
\text{(def)} & \quad \frac{\Gamma, \text{let } x : A = B \vdash^c C : D}{\Gamma \vdash^c (\lambda x : A . C)B : D[x := A]} &
\end{align*}
\]
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 \equiv_{\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}.y') : \alpha[x := y'] = \alpha

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

- Mathematical text in Automath written as a finite list of lines of the form:
  \[ x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T. \]
  Here \( g \) is a new name, an abbreviation for the expression \( t \) of type \( T \) and
  \( x_1, \ldots, x_n \) are the parameters of \( g \), with respective types \( A_1, \ldots, A_n \).

- Each line introduces a new definition which is inherently parametriised by the
  variables occurring in the context needed for it.

- If line \( x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T \) occurs in a book \( \mathcal{B} \) then we
  can unfold the definition by: 
  \[ b(\Sigma_1, \ldots, \Sigma_n) \rightarrow_\delta \Xi_1[x_1, \ldots, x_n := \Sigma_1, \ldots, \Sigma_n]. \]

- Developments of ordinary mathematical theory in Automath (van Benthen
  Jutting) revealed that this combined definition and parameter mechanism is
  vital for keeping proofs manageable and sufficiently readable for humans.
\[ \Delta \Lambda \]

- In AUT-SL, de Bruijn described how a complete AUTOMATH book can be written as a single \( \lambda \)-calculus formula.

- **Disadvantage of AUT-SL:** in order to put the book into the \( \lambda \)-calculus framework, we must first eliminate all definitional lines of the book.

- De Bruijn did not like this: without definitions, formulae grow exponentially.

- For this reason, de Bruijn developed the \( \Delta \Lambda \) with which he wanted to embrace all essential aspects of AUTOMATH apart from type inclusion.

- \( \Delta \Lambda \) is the lambda calculus written in his wagon notation (as above).

- In \( \Delta \Lambda \), de Bruijn favours trees over character strings and does not make use of AT-couples.
Local versus Global reductions

- In $\Delta\Lambda$, de Bruijn replaced $\beta$-reduction by a sequence of local $\beta$-reductions and AT-removals.

- The reason for this is that the delta reductions $\rightarrow_\delta$ of AUTOMATH can be considered as local $\beta$-reductions, and not as ordinary $\beta$-reductions.

- De Bruijn defined local $\beta$-reduction, which keeps the AT-pair and does $\beta$-reduction at one instance (instead of all the instances).

- Example

\[
\langle y \rangle [x] \langle y \rangle x \leftarrow_{L_\beta} \langle y \rangle [x] \langle x \rangle x \rightarrow_{L_\beta} \langle y \rangle [x] \langle x \rangle y
\]

- Doing a further local $\beta$-reduction gives

\[
\langle y \rangle [x] \langle y \rangle y \leftarrow_{L_\beta} \langle y \rangle [x] \langle y \rangle x \leftarrow_{L_\beta} \langle y \rangle [x] \langle x \rangle x \rightarrow_{L_\beta} \langle y \rangle [x] \langle x \rangle y \rightarrow_{L_\beta} \langle y \rangle [x] \langle y \rangle y
\]
• Now we can remove the AT-pair \( \langle y \rangle [x] \) from \( \langle y \rangle [x] \langle y \rangle y \) obtaining \( \langle y \rangle y \).


