λ-calculus à la Automath

Fairouz Kamareddine (Heriot-Watt University)

12 April 2002

Item Notation/Lambda Calculus à la de Bruijn

- $\mathcal{I}$ translates to item notation:

$$\mathcal{I}(x) = x, \qquad \mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B), \qquad \mathcal{I}(A_B) = \langle\mathcal{I}(B)\rangle\mathcal{I}(A)$$

- $(\lambda x.\lambda y.xy)z$ 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 β 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	Item Notation
$(\underline{(\lambda_x.(\lambda_y.\lambda_z.zd)c)b})a$	$\langle a\rangle \underline{\langle b\rangle [x]} \langle c\rangle [y][z]\langle d\rangle z$
$\downarrow_\beta$	$\downarrow_\beta$
$(\underline{(\lambda_y.\lambda_z.zd)c})a$	$(a)\underline{\langle c\rangle [y]}[z]\langle d\rangle z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\lambda_z.zd)a}$	$\underline{\langle a\rangle [z]}(d)z$
$\downarrow_\beta$	$\downarrow_\beta$
ad	$\langle d\rangle a$

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

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* $\overline{s}$ 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 $([y](y)y)[x]x$ the main items are: $([y](y)y)$ and $[x]$.
  $[y]$ and (y) are *not* main items.

- Each main bachelor $[]$ precedes each main bachelor $()$.
  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 *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:
- $\overline{s} \equiv \overbrace{\emptyset}^{\overline{s_0}} \underbrace{[x][y]}_{\overline{s_1}} \overbrace{(a)[z]}^{\overline{s_2}} \underbrace{[x'](b)}_{\overline{s_3}} \overbrace{(c)(d)[y'][z']}^{\overline{s_4}} \underbrace{(e)}_{\overline{s_5}}$

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 $\lambda\sigma$, 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:

bachelor [ ]s	()[ ]-pairs, A_i in CF	bachelor ()s, B_i in CF	end var
$[x_1]\ldots[x_n]$	$(A_1)[y_1]\ldots(A_m)[y_m]$	$(B_1)\ldots(B_p)$	x

- 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

Name	In Classical Notation	In Item Notation
(θ)	$((\lambda_x.N)P)Q$ $\downarrow$ $(\lambda_x.NQ)P$	$(Q)(P)[x]N$ $\downarrow$ $(P)[x](Q)N$
(γ)	$(\lambda_x.\lambda_y.N)P$ $\downarrow$ $\lambda_y.(\lambda_x.N)P$	$(P)[x][y]N$ $\downarrow$ $[y](P)[x]N$
(γ_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

- Regnier [1992] uses term reshuffling and generalized reduction in analyzing perpetual reduction strategies.
- 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

θ-nf:		()[ ]-pairs mixed with bach. [ ]s $(A_1)[x][y][z](A_2)[p]\cdots$	bach. ()s $(B_1)(B_2)\cdots$	end var x
γ-nf:	bach. [ ]s $[x_1][x_2]\cdots$	()[ ]-pairs mixed with bach. ()s $(B_1)(A_1)[x](B_2)\cdots$		end var x
θ-γ-nf:	bach. [ ]s $[x_1][x_2]\cdots$	()[ ]-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. ()s $(B_1)(B_2)\ldots$	end var x
γ-θ-nf:	bach. [ ]s $[x_1][x_2]\cdots$	()[ ]-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. ()s $(B_1)(B_2)\ldots$	end var x

Example

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 θ-nf and γ-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

$$(A_1)[y_1](A_2)[y_2]B \rightarrow_p (A_2)[y_2](A_1)[y_1]B \qquad \text{if } y_1 \notin \mathrm{FV}(A_2)$$

- We define: $[A]$ to be $\{B \mid \theta(\gamma(A)) =_p \theta(\gamma(B))\}$.
- When $B \in [A]$, we write that $B \approx_{\mathtt{equi}} A$.
- $\rightarrow_\theta, \rightarrow_\gamma, =_\gamma, =_\theta, =_p \subset \approx_{\mathtt{equi}} \subset =_\beta$ (strict inclusions).
- Define $\mathrm{CCF}(A)$ as $\{A' \text{ in canonical form} \mid A' =_p \theta(\gamma(A))\}$.

Reduction based on classes [Kamareddine et al., 2001]

- One-step class-reduction $\leadsto_\beta$ is the least compatible relation such that:

$$A \leadsto_\beta B \quad \text{iff} \quad \exists A' \in [A].\exists B' \in [B].\ A' \to_\beta B'$$

- $\leadsto_\beta$ *really acts as reduction on classes:*

- If $A \leadsto_\beta B$ then forall $A' \approx_{\mathtt{equi}} A$, forall $B' \approx_{\mathtt{equi}} B$, we have $A' \leadsto_\beta B'$.

Properties of reduction modulo classes

- *$\rightsquigarrow_\beta$ generalises $\to_g$ and $\to_\beta$:* $\to_\beta \subset \to_g \subset \rightsquigarrow_\beta \subset =_\beta$.
- *$\approx_\beta$ and $=_\beta$ are equivalent:* $A \approx_\beta B$ iff $A =_\beta B$.
- *$\twoheadrightarrow_\beta$ is Church Rosser:*
  If $A \twoheadrightarrow_\beta B$ and $A \twoheadrightarrow_\beta C$, then for some D: $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$.
- *Classes preserve $SN_{\to_\beta}$:* If $A \in SN_{\to_\beta}$ and $A' \in [A]$ then $A' \in SN_{\to_\beta}$.
- *Classes preserve $SN_{\rightsquigarrow_\beta}$:* If $A \in SN_{\rightsquigarrow_\beta}$ and $A' \in [A]$ then $A' \in SN_{\rightsquigarrow_\beta}$.
- *$SN_{\to_\beta}$ and $SN_{\rightsquigarrow_\beta}$ are equivalent:* $A \in SN_{\rightsquigarrow_\beta}$ iff $A \in SN_{\to_\beta}$.

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 $(y)[x]x$.
- $\Pi_{z:*}.(\lambda_{x:z}.x)y$ 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:
- $\mathcal{T} = * \mid \Box \mid V \mid (\mathcal{T}\delta)\mathcal{T} \mid (\mathcal{T}\lambda_V)\mathcal{T} \mid (\mathcal{T}\Pi_V)\mathcal{T}$.
- The typing rules don't change although we do class reduction $\leadsto_\beta$ instead of normal β-reduction $\to_\beta$.
- The typing rules don't change because $=_\beta$ is the same as $\approx_\beta$.

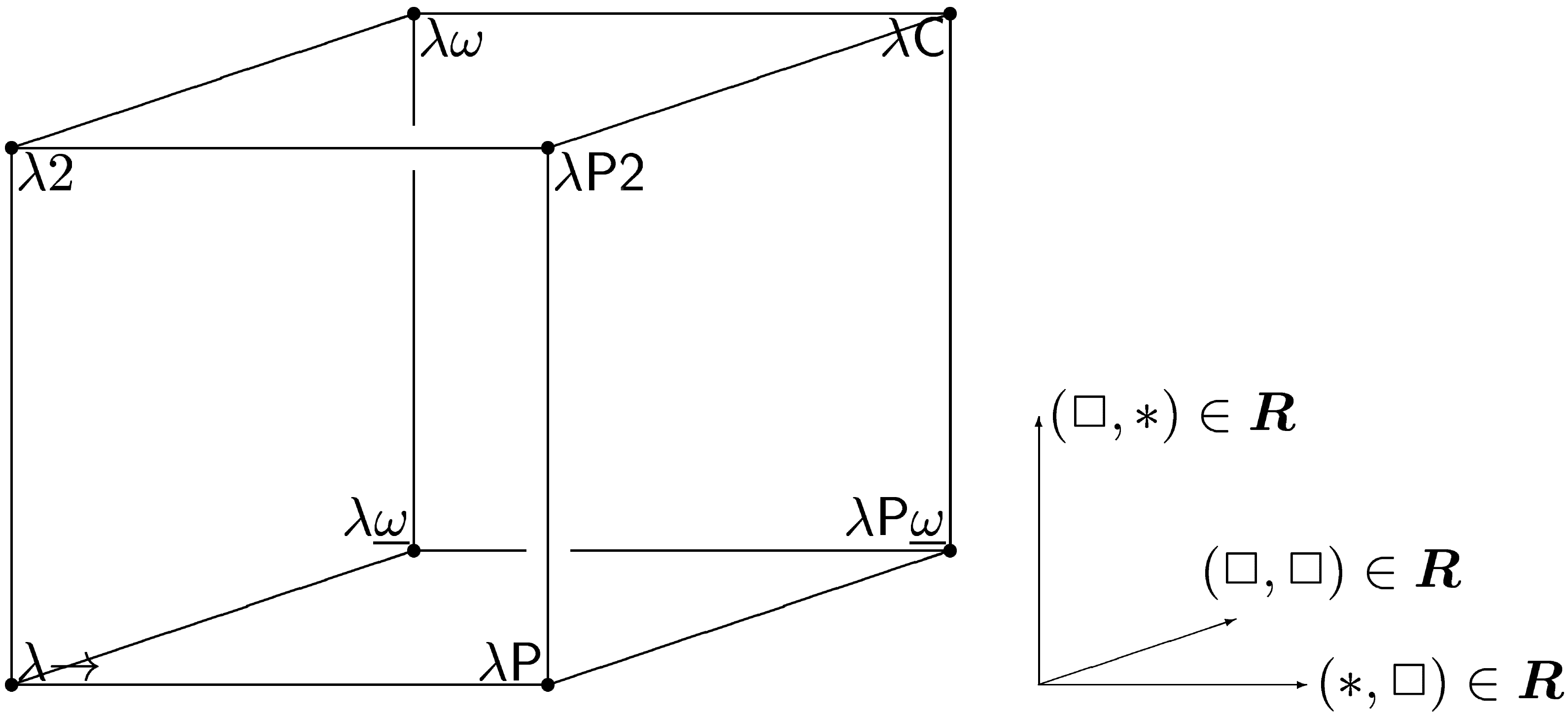

Figure 1: The Barendregt Cube

Subject Reduction fails

- Most properties including SN hold for all systems of the cube extended with class reduction. However, SR only holds in $\lambda_{\to}$ $(*, *)$ and $\lambda\underline{\omega}$ $(\Box, \Box)$.
- SR fails in λP $(*, \Box)$ (and hence in $\lambda P2, \lambda P\underline{\omega}$ and λC). Example in paper.
- SR also fails in $\lambda 2$ $(\Box, *)$ (and hence in $\lambda P2, \lambda\omega$ and λC):

Why does Subject Reduction fails

- $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x \rightsquigarrow_\beta (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x$.
- $(\lambda_{\alpha:*}.\lambda_{y:\alpha}.(\lambda_{x:\alpha}.x)y)\beta y' \rightsquigarrow_\beta (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta$
- $\beta : *, y' : \beta \vdash_{\lambda 2} (\lambda_{\alpha:*}.\lambda_{y:\alpha}.(\lambda_{x:\alpha}.x)y)\beta y' : \beta$
- Yet, $\beta : *, y' : \beta \not\vdash_{\lambda 2} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta : \tau$ for any τ.
- 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: 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 $\Gamma \vdash^{c} \cdot =_{\texttt{def}} \cdot$ to be the equivalence relation generated by:
  - if $A =_{\beta} B$ then $\Gamma \vdash^{c} A =_{\texttt{def}} B$
  - if let $x : M \ = N$ is in Γ and if B arises from A by substituting one particular occurrence of x in A by N, then $\Gamma \vdash^{c} A =_{\texttt{def}} B$.

The (simplified) Cube with definitions and class reduction

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

(start) $\dfrac{\Gamma \vdash^{\mathsf{c}} A : s}{\Gamma, x{:}A \vdash^{\mathsf{c}} x : A}$ $\dfrac{\Gamma \vdash^{\mathsf{c}} A : s \qquad \Gamma \vdash^{\mathsf{c}} B : A}{\Gamma,\ \mathsf{let}\ x : A\ = B \vdash^{\mathsf{c}} x : A}$ x fresh

(weak) $\dfrac{\Gamma \vdash^{\mathsf{c}} D : E \quad \Gamma \vdash^{\mathsf{c}} A : s}{\Gamma, x{:}A \vdash^{\mathsf{c}} D : E}$ $\dfrac{\Gamma \vdash^{\mathsf{c}} A : s \quad \Gamma \vdash^{\mathsf{c}} B : A \quad \Gamma \vdash^{\mathsf{c}} D : E}{\Gamma,\ \mathsf{let}\ x : A\ = B \vdash^{\mathsf{c}} D : E}$ x fresh

(conv) $\dfrac{\Gamma \vdash^{\mathsf{c}} A : B \qquad \Gamma \vdash^{\mathsf{c}} B' : S \qquad \Gamma \vdash^{\mathsf{c}} B =_{\mathtt{def}} B'}{\Gamma \vdash^{\mathsf{c}} A : B'}$

(def) $\dfrac{\Gamma, \mathsf{let}\ x : A\ = B \vdash^{\mathsf{c}} C : D}{\Gamma \vdash^{\mathsf{c}} (\lambda_{x:A}.C)B : D[x := A]}$

Table 1: Definitions solve subject reduction

1.	$\beta : *, y' : \beta,$ let $\alpha : * \ = \beta$	$\vdash^{c} y' : \beta$
2.	$\beta : *, y' : \beta,$ let $\alpha : * = \beta$	$\vdash^{c} \alpha =_{\mathtt{def}} \beta$
3.	$\beta : *, y' : \beta,$ let $\alpha : * \ = \beta$	$\vdash^{c} y' : \alpha$ (from 1 and 2)
4.	$\beta : *, y' : \beta,$ let $\alpha : * \ = \beta,$ let $x : \alpha \ = y'$	$\vdash^{c} x : \alpha$
5.	$\beta : *, y' : \beta,$ let $\alpha : * \ = \beta$	$\vdash^{c} (\lambda_{x:\alpha}.x)y' : \alpha[x := y'] = \alpha$
	$\beta : *, y' : \beta$	$\vdash^{c} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta : \alpha[\alpha := \beta] = \beta$

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 parametrised 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 $\mathfrak{B}$ then we can unfold the definition by: $b(\Sigma_1, \ldots, \Sigma_n) \to_\delta \Xi_1[x_1, \ldots, x_n := \Sigma_1, \ldots, \Sigma_n]$.

- Developments of ordinary mathematical theory in AUTOMATH (van Benthem 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 λ-calculus formula.
- *Disadvantage of* AUT-*SL:* in order to put the book into the λ-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 β-reduction by a sequence of local β-reductions and AT-removals.

- The reason for this is that the delta reductions $\rightarrow_\delta$ of $\textsc{Automath}$ can be considered as local β-reductions, and not as ordinary β-reductions.

- De Bruijn defined local β-reduction, which keeps the AT-pair and does β-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 β-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 $\langle y\rangle y$.

Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pages 233–246, 1995.

Roel Bloo, Fairouz Kamareddine, and Rob Nederpelt. The Barendregt cube with definitions and generalised reduction. *Inform. & Comput.*, 126(2):123–143, May 1996.

Philippe de Groote. The conservation theorem revisited. In *Proc. Int'l Conf. Typed Lambda Calculi and Applications*, pages 163–178. Springer-Verlag, 1993.

F. Kamareddine, R. Bloo, and R. Nederpelt. On Π-conversion in the λ-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1–3):27–45, 1999.

F. Kamareddine, R. Bloo, and R. Nederpelt. De Bruijn's syntax and reductional equivalence of λ-terms. PPDP 2001, ACM publications.

F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-styles of explicit substitutions. *J. Logic Comput.*, 10(3):399–431, 2000.

Fairouz Kamareddine. Postponement, conservation and preservation of strong normalisation for generalised reduction. *J. Logic Comput.*, 10(5):721–738, 2000.

Fairouz Kamareddine and Rob Nederpelt. On stepwise explicit substitution. *Int'l J. Foundations Comput. Sci.*, 4(3):197–240, 1993.

Fairouz Kamareddine and Rob Nederpelt. Refining reduction in the λ-calculus. *J. Funct. Programming*, 5(4):637–651, October 1995.

Fairouz Kamareddine and Rob Nederpelt. A useful λ-notation. *Theoret. Comput. Sci.*, 155(1):85–109, 1996.

Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de Bruijn with explicit substitution. In *7th Int'l Symp. Prog. Lang.: Implem., Logics & Programs, PLILP '95*, volume 982 of *LNCS*, pages 45–62. Springer-Verlag, 1995.

Fairouz Kamareddine and Alejandro Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *J. Funct. Programming*, 7(4):395–420, 1997.

Fairouz Kamareddine, Alejandro Ríos, and J. B. Wells. Calculi of generalised β-reduction and explicit substitutions: The type free and simply typed versions. *J. Funct. Logic Programming*, 1998(5), June 1998.

A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pages 196–207, 1994. ISBN 0-89791-643-3.

A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of β-strong normalization in typed λ-calculi. In *Proc. 10th Ann. IEEE Symp. Logic in Computer Sci.*, pages 311–321, 1995. ISBN 0-8186-7050-9.

Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, March 1994.

Rob Nederpelt. *Strong Normalization in a Typed Lambda Calculus With Lambda Structured Types*. PhD thesis, Eindhoven, 1973.

Laurent Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.