Explicit Extensions in (Typed) λ-calculi

Fairouz Kamareddine

June 2009

Item Notation/Lambda Calculus à la de Bruijn

- $\mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B)$ and $\mathcal{I}(AB) = (\mathcal{I}(B))\mathcal{I}(A)$
- $\mathcal{I}((\lambda x.(\lambda y.xy))z) \equiv (z)[x][y](y)x$. The items are (z), $[x]$, $[y]$ and (y).
- *applicator wagon* (z) and *abstractor wagon* $[x]$ occur NEXT to each other.
- A term is a wagon followed by a term.
- (β) $(\lambda x.A)B \to_\beta A[x := B]$ becomes
- (β) $(B)[x]A \to_\beta A[x := B]$ or $(B)[x]A \to_\beta [x := B]A$
- Sometimes, de Bruijn wrote: (β) $(B)[x]A \to_\beta (B)[x][x := B]A$

Redexes in Item Notation

Classical Notation		*Item Notation*	
$((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$	$\to_\beta$	$(a)(b)[x](c)[y][z](d)z$	$\to_\beta$
$((\lambda_y.\lambda_z.zd)c)a$	$\to_\beta$	$(a)(c)[y][z](d)z$	$\to_\beta$
$(\lambda_z.zd)a$	$\to_\beta ad$	$(a)[z](d)z$	$\to_\beta (d)a$

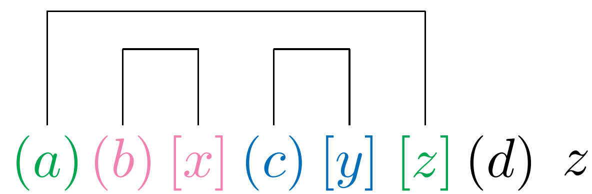


Figure 1: Redexes in item notation

Well-balanced segments

- The "bracketing structure" of $t = ((\lambda_x.(\lambda_y.\lambda_z. - -)c)b)a)$, is compatible with '$\{_1\ \{_2\ \{_3\ \}_2\ \}_1\ \}_3$', where '$\{_i$' and '$\}_i$' match.

- $(a)(b)[x](c)[y][z](d)$ has the bracketing structure $\{\{\ \}\{\ \}\}$.

- Define a well-balanced segment $\overline{s}$ to be a segment of partnered () and [] pairs that match like '$\{$' and '$\}$'.

- Let $\overline{s} \equiv (a)(b)[x](c)[y][z](d)$. Then: (a), (b), $[x]$, (c), $[y]$, and $[z]$, are the *partnered* main items of $\overline{s}$. (d) is a *bachelor* item. $(a)(b)[x](c)[y][z]$ is *well-balanced*.

Generalised reduction

- (general β) $(b)\overline{s}[v]a \rightsquigarrow_\beta \overline{s}\{a[v := b]\}$ if $\overline{s}$ is well-balanced

- Many step general β-reduction $\twoheadrightarrow\!\!\!\!\sim_\beta$ is the reflexive transitive closure of $\rightsquigarrow_\beta$.

- $t \equiv (a)(b)[x](c)[y][z](d)z \rightsquigarrow_\beta$
  $(b)[x](c)[y]\{((d)z)[z := a]\} \equiv$
  $(b)[x](c)[y](d)a$

Lemma 1. *If $a \rightarrow_\beta b$ then $a \rightsquigarrow_\beta b$. And, If $a \rightsquigarrow_\beta b$ then $a =_\beta b$.*

Corollary 1. *If $a \twoheadrightarrow\!\!\!\!\sim_\beta b$ then $a =_\beta b$.* $\square$

Theorem 1. *The general β-reduction is Church-Rosser. I.e. If $a \twoheadrightarrow\!\!\!\!\sim_\beta b$ and $a \twoheadrightarrow\!\!\!\!\sim_\beta c$, then there exists d such that $b \twoheadrightarrow\!\!\!\!\sim_\beta d$ and $c \twoheadrightarrow\!\!\!\!\sim_\beta d$.*

Term reshuffling

- $(a)(b)[x](c)[y][z](d)z$ can be easily rewritten as $(b)[x](c)[y](a)[z](d)z$ by moving the item (a) to the right.

- I.e., we can keep the old β-axiom and we can contract redexes in any order.

- difficult to describe how $((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a$, is rewritten as $(\lambda_x.(\lambda_y.(\lambda_z.zd)a)c)b$.

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

Figure 2: Term reshuffling in item notation

Uses of Generalised reduction and term reshuffling?

- Regnier's *premier redex* in [Reg 92] is a *generalised redex*. [Reg 94] shows that the perpetual reduction strategy finds the longest reduction path when the term is SN. Vidal in [Vid 89] and Sabry in [SF 92] used extended redexes.

- [KTU 94] uses some generalised reduction to show that typability in ML is equivalent to acyclic semi-unification.

- [Nederpelt 73] and [dG 93] and [KW 95a] use generalised reduction and/or term reshuffling to reduce strong normalisation for β-reduction to weak normalisation for related reductions.

- [KW 94] uses amongst other things, generalised reduction and term reshuffling to reduce typability in the rank-2 restriction of system F to the problem of acyclic semi-unification.

- [AFM 95] uses a form of term-reshuffling (which they call "let-C") as a part of an analysis of how to implement sharing in a real language interpreter in a way that directly corresponds to a formal calculus.

- The above description can be found in [KN 95]. Also, [KN 95] showed that generalised reduction makes more redexes visible and hence allows for more flexibility in reducing a term.

- [BKN 96] showed that with generalised reduction one may indeed avoid size explosion without the cost of a longer reduction path and that λ-calculus can be elegantly extended with definitions which result in shorter type derivation.

- [Kam 00] shows that generalised reduction is the first relation for which both conservation and postponement of k-redexes hold. [Kam 00] shows that generalised reduction has PSN.

Partnered and Bachelor Items

"partnered" and "bachelors" items help categorize the main items of a term:

Lemma 2. *Let $\overline{s}$ be the body of a term a. Then the following holds:*

1. *Each bachelor main abstraction item in $\overline{s}$ precedes each bachelor main application item in $\overline{s}$.*
2. *$\overline{s}$ minus all bachelor main items equals a* well-balanced *segment.*
3. *The removal from $\overline{s}$ of all main reducible couples, leaves behind $[v_1]\ldots[v_n](a_1)\ldots(a_m)$, the segment consisting of all bachelor main abstraction and application items.*
4. *If $\overline{s} \equiv \overline{s_1}(b)\overline{s_2}[v]\overline{s_3}$ where $[v]$ and (b) match, then $\overline{s_2}$ is well-balanced.*

Corollary 2. *For each non-empty segment $\overline{s}$, there is a unique* partitioning *in segments $\overline{s_0}, \overline{s_1}, \cdots, \overline{s_n}$, such that $\overline{s} \equiv \overline{s_0}\,\overline{s_1} \cdots \overline{s_n}$ and:*

1. *$\forall 0 \leq i \leq n$, $\overline{s_i}$ is well-balanced in $\overline{s}$ for even i and $\overline{s_i}$ is bachelor in $\overline{s}$ for odd i.*

2. *If $\overline{s_i}$ and $\overline{s_j}$ for $0 \leq i, j \leq n$ are bachelor abstraction resp. application segments, then $\overline{s_i}$ precedes $\overline{s_j}$ in $\overline{s}$.*

3. *If $i \geq 1$ then $\overline{s_{2i}} \not\equiv \emptyset$.* □

This is actually a very nice corollary. It tells us a lot about the structure of our terms.

Example

$\overline{s} \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$, has the following partitioning:

- well-balanced segment $\overline{s_0} \equiv \emptyset$
- bachelor segment $\overline{s_1} \equiv [x][y]$,
- well-balanced segment $\overline{s_2} \equiv (a)[z]$,
- bachelor segment $\overline{s_3} \equiv [x'](b)$,
- well-balanced segment $\overline{s_4} \equiv (c)(d)[y'][z']$,
- bachelor segment $\overline{s_5} \equiv (e)$.

Using () everywhere

- We will replace (a) by $(a\delta)$.
- We will replace $[x]$ by (λ_x) or $(\varepsilon\lambda_x)$; and $[x : A]$ by $(A\lambda_x)$.
- New items: substitution items $(A\sigma_x)$ and typing items $(\Gamma\tau)$.
- For example:

  $(\beta) \qquad (B\delta)(\lambda_x)A \to_\beta (B\delta)(\lambda_x)(B\sigma_x)A$

Type Theory in Item Notation

- $\mathcal{T} = * \mid \Box \mid V \mid \mathcal{T}\mathcal{T} \mid \pi_{V:\mathcal{T}}.\mathcal{T}$

- $(\beta) \qquad (\lambda_{x:B}.A)C \to_\beta A[x := C]$

- $\mathcal{I}$ which translates terms from classical notation to item notation such that:

$$\begin{array}{lcll} \mathcal{I}(A) & = & A & \text{if } A \in \{*, \Box\} \cup V \\ \mathcal{I}(\pi_{x:A}.B) & = & (\mathcal{I}(A)\pi_x)\mathcal{I}(B) & \\ \mathcal{I}(AB) & = & (\mathcal{I}(B)\delta)\mathcal{I}(A) & \end{array}$$

- $(\beta) \qquad (\lambda_{x:B}.A)C \to_\beta A[x := C]$

- $(\beta) \qquad (C\delta)(B\lambda_x)A \to_\beta (C\sigma_x)A$

Trees

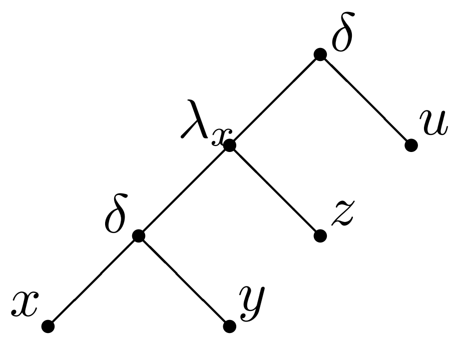


Figure 3: binary tree of $(\lambda_{x:z}.xy)u$

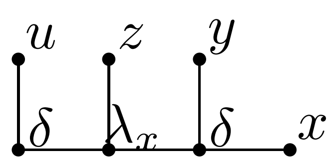


Figure 4: layered tree of $(\lambda_{x:z}.xy)u$

$\mathcal{I}((\lambda_{x:z}.xy)u) \equiv (u\delta)(z\lambda_x)(y\delta)x$

Compatibility

- In Classical notation:
  - $\frac{A_1 \to A_2}{A_1 B \to A_2 B}$ $\frac{B_1 \to B_2}{A B_1 \to A B_2}$
  -
  - $\frac{A_1 \to A_2}{\pi_{x:A_1}.B \to \pi_{x:A_2}.B}$ $\frac{B_1 \to B_2}{\pi_{x:A}.B_1 \to \pi_{x:A}.B_2}$
- In Item notation:
  - $\frac{A_1 \to A_2}{(A_1 \omega) B \to (A_2 \omega) B}$ $\frac{B_1 \to B_2}{(A \omega) B_1 \to (A \omega) B_2}$

Restrictions of terms

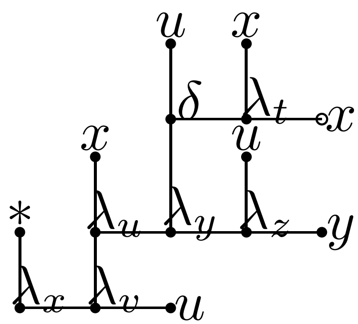
The restriction $t \restriction x^\circ$ of a term t to a variable occurrence x° in t is a term consisting of precisely those "parts" of t that may be relevant for this x°, especially as regards binding, typing and substitution.

- the type of x° in t is the type of x° in $t \restriction x^\circ$,
- the λ's relevant to x° in t appear also in $t \restriction x^\circ$ and have the same binding relation to x°,
- If in t, any substitution for x° is possible, then it is also possible in $t \restriction x^\circ$.

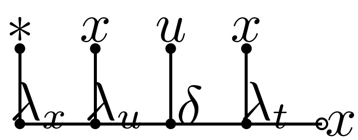
Example of term restriction

- $t \equiv (*\lambda_x)(x\lambda_v)(x\delta)(*\lambda_y)((x\lambda_z)y^\circ\delta)(y\lambda_u)u$.
- Only $(*\lambda_x)$, $(x\lambda_v)$, $(x\delta)$, $(*\lambda_y)$ and $(x\lambda_z)$ are of importance for y°.
  - y° is in the scope of $(*\lambda_x), (x\lambda_v), (*\lambda_y)$ and $(x\lambda_z)$.
  - The x is a candidate for substitution for y°, due to the presence of the $\delta\lambda$-segment $(x\delta)(*\lambda_y)$ meaning that the x will substitute y in $((x\lambda_z)y^\circ\delta)(y\lambda_u)u$.
  - Nothing else in t is relevant to y°.
- $t \upharpoonleft y^\circ$ is $(*\lambda_x)(x\lambda_v)(x\delta)(*\lambda_y)(x\lambda_z)$. Remove everything to the right of y°: $(*\lambda_x)(x\lambda_v)(x\delta)(*\lambda_y)((x\lambda_z)$. Remove all extra parentheses.
- If t is written $\lambda_{x:*}.\lambda_{v:x}.(\lambda_{y:*}.(\lambda_{u:y}.u)\lambda_{z:x}.y^\circ)x$ then $t \upharpoonleft y^\circ$ is less obvious.

restriction trees



$$t \equiv (*\lambda_x)((x\lambda_u)((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y\lambda_v)u$$



$$t\restriction x^\circ \equiv (*\lambda_x)(x\lambda_u)(u\delta)(x\lambda_t)x^\circ$$

Figure 5: A term and its restriction to a variable

Definition of term restriction

Definition 1. $x^\circ \restriction x^\circ \equiv x$ *and* $(t_1\omega)t_2 \restriction x^\circ \equiv \begin{cases} t_1 \restriction x^\circ & \text{if } x^\circ \textit{ occurs in } t_1 \\ (t_1\omega)(t_2 \restriction x^\circ) & \text{if } x^\circ \textit{ occurs in } t_2 \end{cases}$

Let t be $(*\lambda_x)((x\lambda_u)((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y\lambda_v)u$.

$$\begin{aligned}
\text{Then } t \restriction x^\circ &\equiv ((*\lambda_x)((x\lambda_u)((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y\lambda_v)u) \restriction x^\circ \\
&\equiv (*\lambda_x)(((x\lambda_u)((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y\lambda_v)u \restriction x^\circ) \\
&\equiv (*\lambda_x)((x\lambda_u)((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y \restriction x^\circ) \\
&\equiv (*\lambda_x)(x\lambda_u)(((u\delta)(x\lambda_t)x^\circ\lambda_y)(u\lambda_z)y \restriction x^\circ) \\
&\equiv (*\lambda_x)(x\lambda_u)((u\delta)(x\lambda_t)x^\circ \restriction x^\circ) \\
&\equiv (*\lambda_x)(x\lambda_u)(u\delta)((x\lambda_t)x^\circ \restriction x^\circ) \\
&\equiv (*\lambda_x)(x\lambda_u)(u\delta)(x\lambda_t)(x^\circ \restriction x^\circ) \\
&\equiv (*\lambda_x)(x\lambda_u)(u\delta)(x\lambda_t)x
\end{aligned}$$

Describing normal forms in a substitution calculus

[KR 95] provided λs, a calculus of substitution à la de Bruijn, which remains as close as possible to the classical λ-calculus. The set of terms, noted Λs, of the λs-calculus is given as follows:

$$\Lambda s ::= \mathbb{N} \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s\, \sigma^i \Lambda s \mid \varphi^i_k \Lambda s \qquad \textit{where} \quad i \geq 1\,, \quad k \geq 0\,.$$

The set of open terms, noted Λs_{op} is given as follows:

$$\Lambda s_{op} ::= \mathbf{V} \,|\, \mathbb{N} \,|\, \Lambda s_{op} \Lambda s_{op} \,|\, \lambda \Lambda s_{op} \,|\, \Lambda s_{op} \sigma \Lambda s_{op} \,|\, \varphi^i_k \Lambda s_{op} \qquad \textit{where} \quad i \geq 1\,, \quad k \geq 0$$

The λs-calculus

$$
\begin{array}{llcl}
\sigma\text{-generation} & (\lambda a)\, b & \longrightarrow & a\,\sigma^1\, b \\
\sigma\text{-}\lambda\text{-transition} & (\lambda a)\,\sigma^i b & \longrightarrow & \lambda(a\,\sigma^{i+1}\, b) \\
\sigma\text{-app-transition} & (a_1\, a_2)\,\sigma^i b & \longrightarrow & (a_1\,\sigma^i b)\,(a_2\,\sigma^i b) \\
\sigma\text{-destruction} & \mathtt{n}\,\sigma^i b & \longrightarrow & \begin{cases} \mathtt{n-1} & \text{if } n > i \\ \varphi_0^i\, b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i \end{cases} \\
\varphi\text{-}\lambda\text{-transition} & \varphi_k^i(\lambda a) & \longrightarrow & \lambda(\varphi_{k+1}^i\, a) \\
\varphi\text{-app-transition} & \varphi_k^i(a_1\, a_2) & \longrightarrow & (\varphi_k^i\, a_1)\,(\varphi_k^i\, a_2) \\
\varphi\text{-destruction} & \varphi_k^i\,\mathtt{n} & \longrightarrow & \begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \le k \end{cases}
\end{array}
$$

We use λs to denote this set of rules.

The λs_e-calculus

The λs_e-*calculus* is obtained by adding the following rules to those of the λs-calculus.

σ-σ-transition	$(a\sigma b)\,\sigma^j\, c$	$\longrightarrow$	$(a\,\sigma^{j+1}\, c)\,\sigma\,(b\,\sigma^{j-i+1}\, c)$	if	$i \leq j$
σ-φ-transition 1	$(\varphi^i_k\, a)\,\sigma^j\, b$	$\longrightarrow$	$\varphi^{i-1}_k\, a$	if	$k < j < k+i$
σ-φ-transition 2	$(\varphi^i_k\, a)\,\sigma^j\, b$	$\longrightarrow$	$\varphi^i_k(a\,\sigma^{j-i+1}\, b)$	if	$k+i \leq j$
φ-σ-transition	$\varphi^i_k(a\,\sigma^j\, b)$	$\longrightarrow$	$(\varphi^i_{k+1}\, a)\,\sigma^j\,(\varphi^i_{k+1-j}\, b)$	if	$j \leq k+1$
φ-φ-transition 1	$\varphi^i_k\,(\varphi^j_l\, a)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\, a)$	if	$l+j \leq k$
φ-φ-transition 2	$\varphi^i_k\,(\varphi^j_l\, a)$	$\longrightarrow$	$\varphi^{j+i-1}_l\, a$	if	$l \leq k < l+j$

We use λs_e to denote this set of rules.

s_e-normal forms in classical notation

It is cumbersome to describe s_e-normal forms of open terms. But this description is needed to show the weak normalisation of the s_e-calculus. In classical notation, an open term is an s_e-normal form iff one of the following holds:

- $a \in \mathbf{V} \cup \mathbb{N}$, i.e. a is a variable or a de Bruijn number.
- $a = b\,c$, where b and c are s_e-normal forms.
- $a = \lambda b$, where b is an s_e-normal form.
- $a = b\,\sigma^j c$, where c is an s_e-nf and b is an s_e-nf of the form X, or $d\,\sigma^i e$ with $j < i$, or $\varphi^i_k d$ with $j \leq k$.
- $a = \varphi^i_k b$, where b is an s_e-nf of the form X, or $c\,\sigma^j d$ with $j > k+1$, or $\varphi^j_l c$ with $k < l$.

s_e-normal forms in item notation

The s_e-nf's can be described in item notation by the following syntax:

$$NF ::= \mathbf{V} \mid \mathbb{N} \mid (NF\,\delta)NF \mid (\lambda)NF \mid \overline{s}\,\mathbf{V}$$

where $\overline{s}$ is a normal $\sigma\varphi$-segment whose bodies belong to NF. $a\,\sigma^i b = (b\,\sigma^i)a$ and $\varphi^i_k a = (\varphi^i_k)a$. $(b\,\sigma^i)$ and (φ^i_k) are called σ- and φ-items respectively. b and a are the *bodies* of these respective items.

A *normal $\sigma\varphi$-segment* $\overline{s}$ is a sequence of σ- and φ-items such that every pair of adjacent items in $\overline{s}$ are of the form:

$(\varphi^i_k)(\varphi^j_l)$ and $k < l$
$(b\,\sigma^i)(c\,\sigma^j)$ and $i < j$

$(\varphi^i_k)(b\,\sigma^j)$ and $k < j - 1$
$(b\,\sigma^j)(\varphi^i_k)$ and $j \leq k$.

Types

- At the end of the nineteenth century, types did not play a role in mathematics or logic, unless at the meta-level, in order to distinguish between different ‘classes’ of objects.

- Frege’s formalization of logical reasoning, as explained in the *Begriffsschrift* ([Frege 1879]), was untyped.

- Only after the discovery of Russell’s paradox, undermining Frege’s work, one may observe various formulations of typed theories.

- The first theory which aimed at avoiding the paradoxes using types was that of Russell and Whitehead, as exposed in their famous *Principia Mathematica* ([Whitehead and Russell 1910]).

- Church developed a theory of functionals which is nowadays called λ-*calculus* ([Church 1932]).

- This calculus was used for defining a notion of computability that turned out to be of the same power as Turing-computability or general recursiveness.

- However, the original, untyped version did not work as a foundation for mathematics.

- In order to come round the inconsistencies in his proposal for logic, Church developed the 'simple theory of types' $\lambda_{\rightarrow}$ ([Church 1940]).

- From then till the present day, research on the area has grown and one can find various reformulations of type theories.

- A taxonomy of type systems has been given by Barendregt ([Bar 92]).

- Church's $\lambda_{\rightarrow}$ is the lowest system on the Cube.

- $\lambda_{\to}$ has, apart from *type variables*, so-called *arrow-types* of the form $A \to B$.

- In higher type theories, arrow-types are replaced by dependent products $\Pi_{x:A}.B$, where B may contain x as a free variable, and thus may *depend on* x. Example: $\Pi_{A:*}.\lambda_{x:A}.x$

- This means that abstraction can be over types, similarly to the abstraction over terms: $\lambda_{x:A}.b$.

Barendregt Cube

(axiom) $<>\vdash_\beta * : \Box$

(start rule) $$\frac{\Gamma \vdash_\beta A : S}{\Gamma.\lambda_{x:A} \vdash_\beta x : A}\ x \notin \Gamma$$

(weakening rule) $$\frac{\Gamma \vdash_\beta A : S \qquad \Gamma \vdash_\beta D : E}{\Gamma.\lambda_{x:A} \vdash_\beta D : E}\ x \notin \Gamma$$

(application rule) $$\frac{\Gamma \vdash_\beta F : \Pi_{x:A}.B \qquad \Gamma \vdash_\beta a : A}{\Gamma \vdash_\beta Fa : B[x := a]}$$

(abstraction rule) $$\frac{\Gamma.\lambda_{x:A} \vdash_\beta b : B \qquad \Gamma \vdash_\beta \Pi_{x:A}.B : S}{\Gamma \vdash_\beta \lambda_{x:A}.b : \Pi_{x:A}.B}$$

(conversion rule) $$\frac{\Gamma \vdash_\beta A : B \qquad \Gamma \vdash_\beta B' : S \qquad B =_\beta B'}{\Gamma \vdash_\beta A : B'}$$

(formation rule) $$\frac{\Gamma \vdash_\beta A : S_1 \qquad \Gamma.\lambda_{x:A} \vdash_\beta B : S_2}{\Gamma \vdash_\beta \Pi_{x:A}.B : S_2}\ \text{if } (S_1, S_2) \text{ is a rule}$$

System	*Allowed* (S_1, S_2) *rules*			
$\lambda_{\to}$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

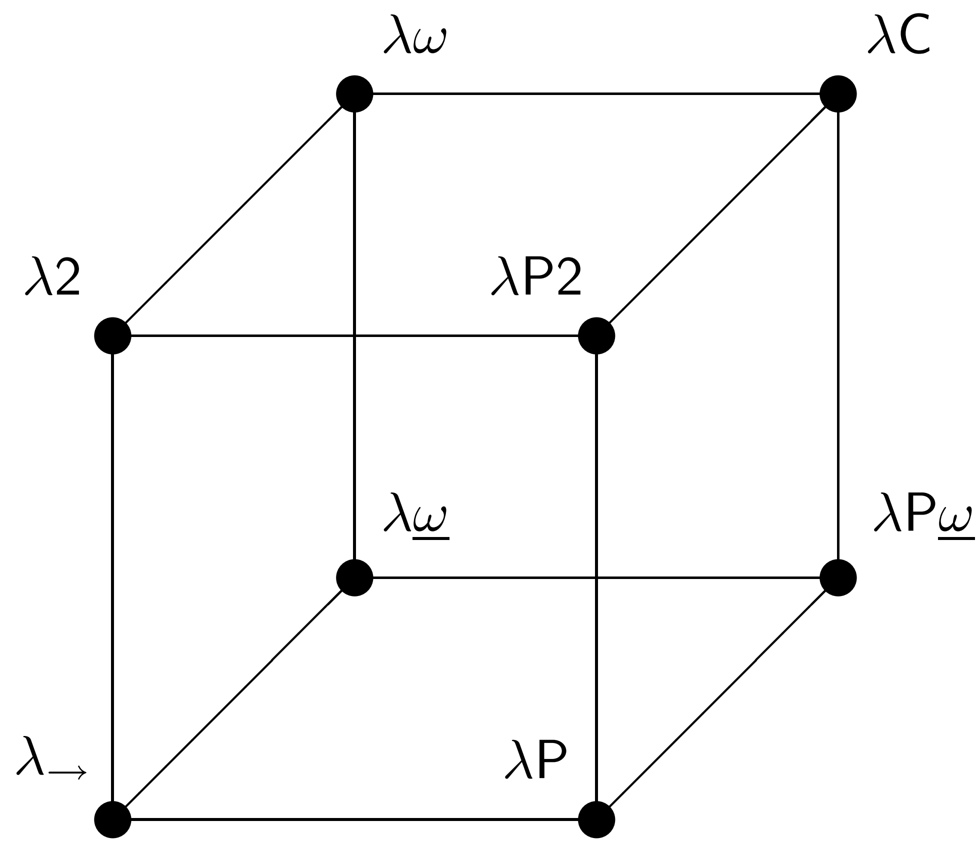


Figure 6: The Cube

Example derivation

Take $\Gamma \equiv \lambda_{\beta:*}.\lambda_{y:\beta}$. In $\lambda 2$, using the rules $(*,*)$ and $(\Box,*)$ we have:

$\Gamma \vdash_{\lambda 2} y : \beta : * : \Box$	
$\Gamma.\lambda_{\alpha:*} \vdash_{\lambda 2} \alpha : *$	(start)
$\Gamma.\lambda_{\alpha:*}.\lambda_{x:\alpha} \vdash_{\lambda 2} x : \alpha : *$	(start resp weakening)
$\Gamma.\lambda_{\alpha:*} \vdash_{\lambda 2} \Pi_{x:\alpha}.\alpha : *$	(formation rule $(*,*)$)
$\Gamma.\lambda_{\alpha:*} \vdash_{\lambda 2} \lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha$	(abstraction)
$\Gamma \vdash_{\lambda 2} \Pi_{\alpha:*}.\Pi_{x:\alpha}.\alpha : *$	(formation rule $(\Box,*)$)
$\Gamma \vdash_{\lambda 2} \lambda_{\alpha:*}.\lambda_{x:\alpha}.x : \Pi_{\alpha:*}\Pi_{x:\alpha}.\alpha$	(abstraction)
$\Gamma \vdash_{\lambda 2} (\lambda_{\alpha:*}.\lambda_{x:\alpha})\beta : \Pi_{x:\beta}.\beta$	(application, we already knew $\Gamma \vdash_{\lambda 2} \beta : *$)
$\Gamma \vdash_{\lambda 2} (\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta y : \beta$	(application, we already knew $\Gamma \vdash_{\lambda 2} y : \beta$)

It is not possible to derive this judgement in $\lambda_{\rightarrow}$ as $(\Box,*)$ is needed.

The system $\lambda_{\rightarrow}$

(axiom) $$<> \vdash_\beta * : \Box$$

(start rule) $$\frac{\Gamma \vdash_\beta A : S}{\Gamma.\lambda_{x:A} \vdash_\beta x : A} \; x \notin \Gamma$$

(weakening rule) $$\frac{\Gamma \vdash_\beta A : S \qquad \Gamma \vdash_\beta D : E}{\Gamma.\lambda_{x:A} \vdash_\beta D : E} \; x \notin \Gamma$$

(application rule) $$\frac{\Gamma \vdash_\beta F : A \rightarrow B \qquad \Gamma \vdash_\beta a : A}{\Gamma \vdash_\beta Fa : B}$$

(abstraction rule) $$\frac{\Gamma.\lambda_{x:A} \vdash_\beta b : B \qquad \Gamma \vdash_\beta A \rightarrow B : S}{\Gamma \vdash_\beta \lambda_{x:A}.b : A \rightarrow B}$$

(conversion rule) $$\frac{\Gamma \vdash_\beta A : B \qquad \Gamma \vdash_\beta B' : S \qquad B =_\beta B'}{\Gamma \vdash_\beta A : B'}$$

(formation rule) $$\frac{\Gamma \vdash_\beta A : * \qquad \Gamma.\lambda_{x:A} \vdash_\beta B : *}{\Gamma \vdash_\beta \Pi_{x:A}.B : *}$$

The system $\lambda_\rightarrow$ revised

(start rule) $\dfrac{\Gamma \vdash_\beta A : S}{\Gamma.\lambda_{x:A} \vdash_\beta x : A}\; x \notin \Gamma$

(weakening rule) $\dfrac{\Gamma \vdash_\beta A : S \qquad \Gamma \vdash_\beta D : E}{\Gamma.\lambda_{x:A} \vdash_\beta D : E}\; x \notin \Gamma$

(application rule) $\dfrac{\Gamma \vdash_\beta F : A \rightarrow B \qquad \Gamma \vdash_\beta a : A}{\Gamma \vdash_\beta Fa : B}$

(abstraction rule) $\dfrac{\Gamma.\lambda_{x:A} \vdash_\beta b : B}{\Gamma \vdash_\beta \lambda_{x:A}.b : A \rightarrow B}$

Desirable Properties: See [Bar 92]

If $\Gamma \vdash A : B$ then A and B are legal *expressions* and Γ is a legal *context*.

Theorem 2. *(The Church Rosser Theorem CR, for $\twoheadrightarrow_\beta$) If $A \twoheadrightarrow_\beta B$ and $A \twoheadrightarrow_\beta C$ then there exists D such that $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$*

Lemma 3. *Correctness of types for $\vdash_\beta$)*
If $\Gamma \vdash_\beta A : B$ then $(B \equiv \Box$ or $\Gamma \vdash_\beta B : S$ for some sort $S)$.

Theorem 3. *(Subject Reduction SR, for $\vdash_\beta$ and $\twoheadrightarrow_\beta$)*
If $\Gamma \vdash_\beta A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash_\beta A' : B$

Theorem 4. *(Strong Normalisation with respect to $\vdash_\beta$ and $\twoheadrightarrow_\beta$)*
For all $\vdash_\beta$-legal terms M, we have $SN_{\twoheadrightarrow_\beta}(M)$. I.e. M is strongly normalising with respect to $\twoheadrightarrow_\beta$.

Π-reduction: See [KN 96a]

- Once we allow abstraction over types, it would be nice to discuss the reduction rules which govern these types.

- We want: $(\lambda_{x:A}.b)C \to_\beta b[x := C]$, as well as $(\Pi_{x:A}.B)C \to_\beta B[x := C]$.

- This strategy of permitting Π-*application* $(\Pi_{x:A}.B)C$ in term construction is not commonly used, yet is desirable for the following reasons:

- (2) below is more elegant and uniform than (1).

$$\text{If } f : \Pi_{x:A}.B \text{ and } a : A, \text{ then } fa : B[x := a] \tag{1}$$

$$\text{If } f : \Pi_{x:A}.B \text{ and } a : A, \text{ then } fa : (\Pi_{x:A}.B)a. \tag{2}$$

- With Π-reduction, one introduces a *compatibility property* for the typing of applications:

$$M : N \Rightarrow MP : NP.$$

This is in line with the compatibility property for the typing of abstractions, which *does* hold in general:

$$M : N \Rightarrow \lambda_{y:P} M : \Pi_{y:P} N.$$

- $$\begin{array}{llll} A : *, b : A, a : A & \vdash & a : A & \text{(start)} \\ A : *, b : A & \vdash & (\lambda_{a:A}.a) : (\Pi_{a:A}.A) & \text{(abstraction)} \\ A : *, b : A & \vdash & (\lambda_{a:A}.a)b : (\Pi_{a:A}.A)b & \text{(application)} \\ A : *, b : A & \vdash & (\lambda_{a:A}.a)b : A & \text{(conversion)} \end{array}$$

- The ability to divide two important questions of typing. $\Gamma \vdash A : B$ becomes:
  1. Is A typable in Γ? $\Gamma \vdash A$.
  2. Is B the type of A in Γ? How does $\tau(\Gamma, A)$ and B compare.

- In a compiler, Π-reduction allows to separate the type finder from the evaluator since $\vdash$ no longer mentions substitution. One first extracts the type and only then evaluates it.

- This is related to some work of Peyton-Jones in his language Henk.

Extending the Cube with Π-reduction: See [KN 96a]

$\beta\Pi$-reduction $\rightarrow_{\beta\Pi}$, is the least compatible relation generated out of the following axiom:

$$(\beta\Pi) \qquad (\pi_{x:B}.A)C \rightarrow_{\beta\Pi} A[x := C]$$

$\twoheadrightarrow_{\beta\Pi}$ is the reflexive transitive closure of $\rightarrow_{\beta\Pi}$. $=_{\beta\Pi}$ is the least equivalence relation generated by $\twoheadrightarrow_{\beta\Pi}$.

(new application rule) $$\frac{\Gamma \vdash_{\beta\Pi} F : \Pi_{x:A}.B \qquad \Gamma \vdash_{\beta\Pi} a : A}{\Gamma \vdash_{\beta\Pi} Fa : (\Pi_{x:A}.B)a}$$

(new conversion rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : B \qquad \Gamma \vdash_{\beta\Pi} B' : S \qquad B =_{\beta\Pi} B'}{\Gamma \vdash_{\beta\Pi} A : B'}$$

Barendregt Cube with Π-reduction

(axiom) $<> \vdash_{\beta\Pi} * : \Box$

(start rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : S}{\Gamma.\lambda_{x:A} \vdash_{\beta\Pi} x : A} \; x \notin \Gamma$$

(weakening rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : S \qquad \Gamma \vdash_{\beta\Pi} D : E}{\Gamma.\lambda_{x:A} \vdash_{\beta\Pi} D : E} \; x \notin \Gamma$$

(new application rule) $$\frac{\Gamma \vdash_{\beta\Pi} F : \Pi_{x:A}.B \qquad \Gamma \vdash_{\beta\Pi} a : A}{\Gamma \vdash_{\beta\Pi} Fa : (\Pi_{x:A}.B)a}$$

(abstraction rule) $$\frac{\Gamma.\lambda_{x:A} \vdash_{\beta\Pi} b : B \qquad \Gamma \vdash_{\beta\Pi} \Pi_{x:A}.B : S}{\Gamma \vdash_{\beta\Pi} \lambda_{x:A}.b : \Pi_{x:A}.B}$$

(new conversion rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : B \qquad \Gamma \vdash_{\beta\Pi} B' : S \qquad B =_{\beta\Pi} B'}{\Gamma \vdash_{\beta\Pi} A : B'}$$

(formation rule) $$\frac{\Gamma \vdash_{\beta\Pi} A : S_1 \qquad \Gamma.\lambda_{x:A} \vdash_{\beta\Pi} B : S_2}{\Gamma \vdash_{\beta\Pi} \Pi_{x:A}.B : S_2}$$ if (S_1, S_2) is a rule

Generation Lemma

Lemma 4. *(Generation Lemma for $\vdash_\beta$)*

- *If* $\Gamma \vdash_\beta \Pi_{x:A}.B : C$ *then* $\Gamma \vdash_\beta A : S_1$ *and* $\Gamma.\lambda_{x:A} \vdash_\beta B : S_2$ *and* (S_1, S_2) *is a rule,* $C =_\beta S_2$ *and.....*

- *If* $\Gamma \vdash_\beta Fa : C$ *then* $\Gamma \vdash_\beta F : \Pi_{x:A}.B$ *and* $\Gamma \vdash_\beta a : A$ *and* $C =_\beta B[x := a]$ *and .....*

- *..................*

In Generation lemma for $\vdash_{\beta\Pi}$ for application case, we replace $B[x := a]$ by $(\Pi_{x:A}.B)a$ and change β to to $\beta\Pi$ everywhere.

Correctness of types fails for Π-reduction even in $\lambda_{\to}$

Lemma 5. *For any* A, B, C, S *we have* $\Gamma \not\vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$.

Proof: *If* $\Gamma \vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$ *then by generation*, $\Gamma \vdash_{\beta\Pi} \Pi_{x:A}.B : \Pi_{x:A'}.B'$ *and again by generation*, $\Gamma.\lambda_{x:A} \vdash_{\beta\Pi} B : S' \wedge S' =_{\beta\Pi} \Pi_{x:A'}.B'$. *Absurd.* $\Box$

The new legal terms of the form $(\Pi_{x:B}.C)A$ imply the failure of Correctness of types Lemma 3 for $\vdash_{\beta\Pi}$ even in $\lambda_{\to}$.

- $\Gamma \vdash_{\beta\Pi} A : B$ may not imply $B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some sort S.
- E.g., if $\Gamma \equiv \lambda_{z:*}.\lambda_{x:z}$ then $\Gamma \vdash_{\beta\Pi} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$, but $\Gamma \not\vdash_{\beta\Pi} (\Pi_{y:z}.z)x : S$ from Lemma 5.

We have a weak correctness of types:

Lemma 6. *If $\Gamma \vdash_{\beta\Pi} A : B$ and B is not a Π-redex then $(B \equiv \Box$ or $\Gamma \vdash_{\beta\Pi} B : S$ for some sort $S)$.*

Proof: *By a trivial induction on the derivation of $\Gamma \vdash_{\beta\Pi} A : B$ noting that the application rule does not apply as $(\Pi_{x:A}.B)a$ is not a Π-redex.* $\Box$

Failure of correctness of types implies failure of Subject Reduction even in $\lambda_{\to}$:

- In $\lambda_{\to}$, we have: $\lambda_{z:*}.\lambda_{x:z} \not\vdash_{\beta\Pi} x : (\Pi_{y:z}.z)x$.
- Otherwise,by generation: $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} (\Pi_{y:z}.z)x : S$, which is absurd by Lemma 5.
- Yet in $\lambda_{\to}$, we have: $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.

Relating $\vdash_{\beta\Pi}$ and $\vdash_\beta$ and Weak SR

For A $\vdash_{\beta\Pi}$-legal, let $\hat{A}$ be $C[x := D]$ if $A \equiv (\Pi_{x:B}.C)D$ and A otherwise.

Lemma 7.

1. *If* $\Gamma \vdash_{\beta\Pi} A : B$ *then* $\Gamma \vdash_\beta A : \hat{B}$*.*

2. *If* $\Gamma \vdash_\beta A : B$ *then* $\Gamma \vdash_{\beta\Pi} A : B$*.*

Lemma 8. *(Weak Subject Reduction for* $\vdash_{\beta\Pi}$ *and* $\twoheadrightarrow_{\beta\Pi}$*)*

1. *If* $\Gamma \vdash_{\beta\Pi} A : B$ *and* $A \twoheadrightarrow_{\beta\Pi} A'$ *then* $\Gamma \vdash_{\beta\Pi} A' : \hat{B}$

2. *If* $\Gamma \vdash_{\beta\Pi} A : B$ *and* $A \twoheadrightarrow_{\beta\Pi} A'$ *and* B *is* $\vdash_\beta$*-legal then* $\Gamma \vdash_{\beta\Pi} A' : B$

Canonical typing

There are reasons why separating the questions "what is the type of a term" (via τ) and "is the term typable" (via $\vdash$), is advantageous. Here are some:

- The canonical type of A is easy to calculate.
- $\tau(A)$ plays the role of a preference type for A. The preference type of $A \equiv \lambda_{x:*}.(\lambda_{y:*}.y)x$ is $\tau(<>, A) \equiv \Pi_{x:*}.(\Pi_{y:*}.*)x$ which $\twoheadrightarrow_{\beta\Pi}$ to $\Pi_{y:*}.*$, the type traditionally given to A.
- The conversion rule is no longer needed as a separate rule in the definition of $\vdash$. It is accommodated in our application rule:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash AB} \text{ if } \tau(\Gamma, A) =_{\beta\Pi} \Pi_{x:C}.D \text{ and } \tau(\Gamma, B) =_{\beta\Pi} C$$

It will be the case that $\tau(\Gamma, AB) \equiv \tau(\Gamma, A)B =_{\beta\Pi} (\Pi_{x:C}.D)B \to_{\beta\Pi} D[x := B]$ and so indeed $\tau(\Gamma, AB) =_{\beta\Pi} D[x := C]$.

- Higher degrees:If we use λ^1 for Π and λ^2 for λ then we can aim for a possible generalization. In fact, we can extend our system by incorporating more different λ's. For example, with an infinity of λ's, viz. λ^0, λ^1, λ^2, λ^3 ..., we replace $\tau(\Gamma, \lambda_{x:A}.B) \equiv \Pi_{x:A}.\tau(\Gamma.\lambda_{x:A}, B)$ and $\tau(\Gamma, \Pi_{x:A}.B) \equiv \tau(\Gamma.\lambda_{x:A}, B)$ by the following:

$$\tau(\Gamma, \lambda^{i+1}_{x:A}.B) \equiv \lambda^{i}_{x:A}.\tau(\Gamma.\lambda_{x:A}, B), \text{for } i = 0, 1, 2, \dots \text{ where } \lambda^0_{x:A}.B \equiv B$$

  There may be circumstances in which one desires to have more "layers" of λ's. As an example we refer to [de Bruijn 74].

- This notion enables one to separate the judgement $\Gamma \vdash A : B$ in two: $\Gamma \vdash A$ and $\tau(\Gamma, A) = B$.

$$
\begin{array}{lcll}
\tau(\Gamma, *) & \equiv & \Box & \\
\tau(\Gamma, x) & \equiv & A \text{ if } (A\lambda_x) \in \Gamma & \\
\tau(\Gamma, (a\delta)F) & \equiv & (a\delta)\tau(\Gamma, F) & \\
\tau(\Gamma, (A\lambda_x)B) & \equiv & (A\Pi_x)\tau(\Gamma(A\lambda_x), B) & \text{if } x \notin dom(\Gamma) \\
\tau(\Gamma, (A\Pi_x)B) & \equiv & \tau(\Gamma(A\lambda_x), B) & \text{if } x \notin dom(\Gamma)
\end{array}
$$

- In usual type theory:
  - the type of $(*\lambda_x)(x\lambda_y)y$ is $(*\Pi_x)(x\Pi_y)x$ and
  - the type of $(*\Pi_x)(x\Pi_y)x$ is $*$.

- With our τ, we get the same result:
  - $\tau(<>, (*\lambda_x)(x\lambda_y)y) \equiv (*\Pi_x)\tau((*\lambda_x), (x\lambda_y)y) \equiv (*\Pi_x)(x\Pi_y)\tau((*\lambda_x)(x\lambda_y), y$
    $(*\Pi_x)(x\Pi_y)x$ and
  - $\tau(<>, (*\Pi_x)(x\Pi_y)x) \equiv \tau((*\lambda_x), (x\Pi_y)x) \equiv \tau((*\lambda_x)(x\lambda_y), x) \equiv *$

Let $\Gamma_0 \equiv <>$, $\Gamma_1 \equiv (*\lambda_z)$, $\Gamma_2 \equiv (*\lambda_z)(*\lambda_y)$, $\Gamma_3 \equiv \Gamma_2(*\lambda_x)$. We want to find the canonical type of $(*\Pi_z)(B\delta)(*\lambda_y)(y\delta)(*\lambda_x)x$ in Γ_0.

$(\Gamma_0\tau)$	$(*\Pi_z)$		$(B\delta)$		$(*\lambda_y)$		$(y\delta)$		$(*\lambda_x)$	
		$(\Gamma_1\tau)$	$(B\delta)$		$(*\lambda_y)$		$(y\delta)$		$(*\lambda_x)$	
			$(B\delta)$	$(\Gamma_1\tau)$	$(*\lambda_y)$		$(y\delta)$		$(*\lambda_x)$	
			$(B\delta)$		$(*\Pi_y)$	$(\Gamma_2\tau)$	$(y\delta)$		$(*\lambda_x)$	
			$(B\delta)$		$(*\Pi_y)$		$(y\delta)$	$(\Gamma_2\tau)$	$(*\lambda_x)$	
			$(B\delta)$		$(*\Pi_y)$		$(y\delta)$		$(*\Pi_x)$	$(\Gamma_3\tau)$
			$(B\delta)$		$(*\Pi_y)$		$(y\delta)$		$(*\Pi_x)$	

New Typability

(⊢-axiom) $<> \vdash *$

(⊢-start rule) $\dfrac{\Gamma \vdash A}{\Gamma(A\lambda_x) \vdash x}$ if vc

(⊢-weakening rule) $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash D}{\Gamma(A\lambda_x) \vdash D}$ if vc

(⊢-application rule) $\dfrac{\Gamma \vdash F \qquad \Gamma \vdash a}{\Gamma \vdash (a\delta)F}$ if ap

(⊢-abstraction rule) $\dfrac{\Gamma(A\lambda_x) \vdash b \qquad \Gamma \vdash (A\Pi_x)B}{\Gamma \vdash (A\lambda_x)b}$ if ab

(⊢-formation) $\dfrac{\Gamma \vdash A \qquad \Gamma(A\lambda_x) \vdash B}{\Gamma \vdash (A\Pi_x)B}$ if fc

- vc (variable condition): $x \not\in \Gamma$ and $\tau(\Gamma, A) \twoheadrightarrow_{\beta\Pi} S$ for some S

- ap (application condition): $\tau(\Gamma, F) =_{\beta\Pi} (A\Pi_x)B$ and $\tau(\Gamma, a) =_{\beta\Pi} A$ for some A, B.

- ab (abstraction condition): $\tau(\Gamma(A\lambda_x), b) =_{\beta\Pi} B$ and $\tau(\Gamma, (A\Pi_x)B) \twoheadrightarrow_{\beta\Pi} S$ for some S.

- fc (formation condition): $\tau(\Gamma, A) \twoheadrightarrow_{\beta\Pi} S_1$ and $\tau(\Gamma(A\lambda_x), B) \twoheadrightarrow_{\beta\Pi} S_2$ for some rule (S_1, S_2).

Properties of $\vdash$

Define $\overline{A}$ to be the $\beta\Pi$-normal form of A.

Lemma 9. *If* $\Gamma \vdash A$ *then* $\downarrow \overline{\tau(\Gamma, A)}$ *and* $\Gamma \vdash_\beta A : \overline{\tau(\Gamma, A)}$

Lemma 10. *(Subject Reduction for* $\vdash$ *and* τ*)*
$\Gamma \vdash A \wedge A \twoheadrightarrow_{\beta\Pi} A' \Rightarrow [\Gamma \vdash A' \wedge \tau(\Gamma, A) =_{\beta\Pi} \tau(\Gamma, A')]$

Theorem 5. *(Strong Normalisation for* $\vdash$*)*
If A *is* $\Gamma^{\vdash}$*-legal, then* $SN_{\twoheadrightarrow_\beta}(A)$.

Lemma 11. $\Gamma \vdash_\beta A : B \iff \Gamma \vdash A$ *and* $\tau(\Gamma, A) =_{\beta\Pi} B$ *and* B *is* $\vdash_\beta$*-legal type.*

Properties of the Cube with generalised reduction

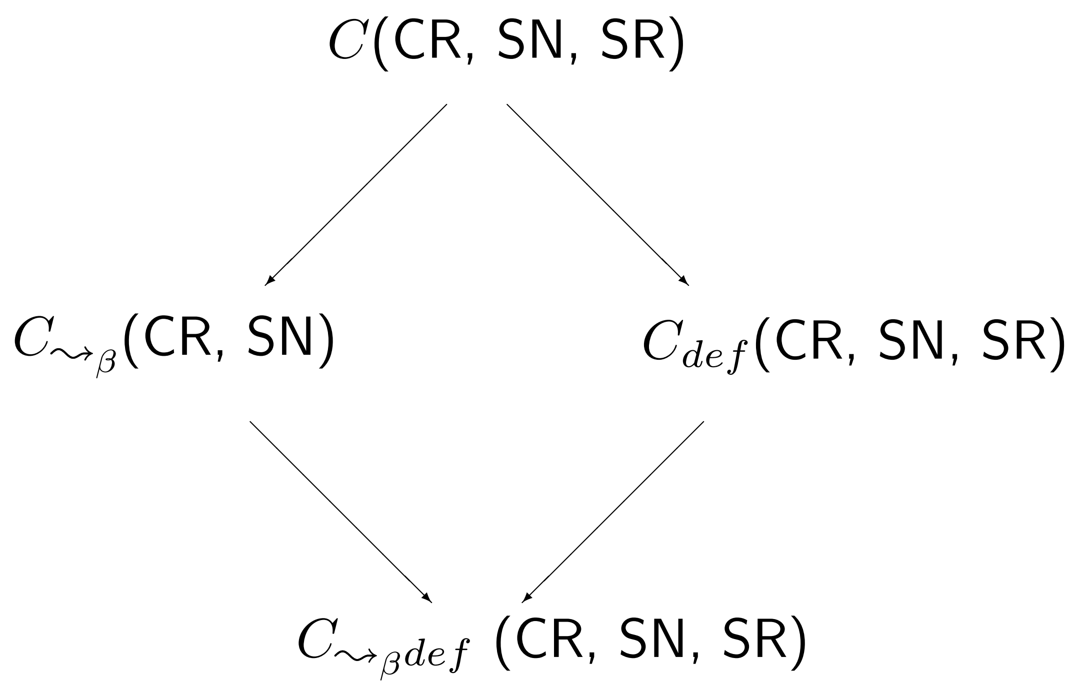


Figure 7: Properties of the Cube with generalised reduction

References

[AFM 95] Ariola, Z.M. Felleisen, M. Maraist, J. Odersky, M. and Wadler, P., A call by need lambda calculus, *Conf. Rec. 22nd Ann. ACM Symp. Princ. Program. Lang. ACM*, 1995.

[Bar 84] Barendregt, H., *Lambda Calculus: its Syntax and Semantics,* North-Holland, 1984.

[Bar 92] Barendregt, H., Lambda calculi with types, *Handbook of Logic in Computer Science*, volume II, ed. Abramsky S., Gabbay D.M., Maibaum T.S.E., Oxford University Press, 1992.

[de Bruijn 74] Bruijn, N.G. de, Some extensions of AUTOMATH: the AUT-4 family, Dept. of Mathematics, Eindhoven University of Technology, 1974.

[BKN 96] Bloo, R., Kamareddine, F., Nederpelt, R. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation 126(2)*, 123-143, 1996.

[dG 93] de Groote, P., The conservation theorem revisited, *Int'l Conf. Typed Lambda Calculi and Applications*, vol. 664 of LNCS, 163-178, Springer-Verlag, 1993.

[KN 95] Kamareddine, F., and Nederpelt, R.P., Generalising reduction in the λ-calculus, *Journal of Functional Programming 5 (4)*, 637-651, 1995.

[KN 96a] F. Kamareddine and R. Nederpelt, Canonical Typing and π-conversion in the Barendregt Cube, *Functional Programming 6*, 1996.

[Church 1932] Church, A., A set of postulates for the foundation of logic, *Annals of Math.* 33 (1932), 346–366 and 34 (1933), 839–864.

[Church 1940] Church, A., A formulation of the simple theory of types, *Journal of Symbolic Logic* 5 (1940), 56–68.

[Frege 1879] Frege, G., *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* (Halle, Verlag von Louis Nebert, 1879). Reprint 1964 (Hildesheim, Georg Olms Verlagsbuchhaltung).

[KN 96b] Kamareddine, F., and Nederpelt, R.P., A useful λ-notation, *Theoretical Computer Science 155*, 1996.

[KR 95] Kamareddine, F., and Rios, A., λ-calculus à la de Bruijn & explicit substitution, *Lecture Notes in Computer Science 982*, 7th international symposium on Programming Languages: Implementations, Logics and Programs, PLILP '95, 45-62, Springer-Verlag, 1995.

[KR 96] Kamareddine, F., and Rios, A., Generalised β_e-reduction and explicit substitution, Computing Science Research Report, University of Glasgow, 1996.

[Kam 00] Kamareddine, F., A reduction relation for which postponement of K-contractions, Conservation and Preservation of Strong Normalisation hold, *Journal of Logic and Computation*, volume 10, number 5, 2000.

[KTU 94] Kfoury, A.J., Tiuryn, J. and Urzyczyn, P., An analysis of ML typability, J. ACM 41(2), 368-398, 1994.

[KW 94] Kfoury, A.J. and Wells, J.B., A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus, *Proc. 1994 ACM Conf. LISP Funct. Program.*, 1994.

[KW 95a] Kfoury, A.J. and Wells, J.B., New notions of reductions and non-semantic proofs of β-strong normalisation in typed λ-calculi, *LICS*, 1995.

[KW 95b] Kfoury, A.J. and Wells, J.B., Addendum to new notions of reduction and non-semantic proofs of β-strong normalisation in typed λ-calculi, Boston University.

[BLR 95] Benaissa, Briaud, Lescanne, Rouyer-Degli, $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation, personel communication, 1995.

[MN 95] Muñoz C., Confluence and preservation of strong normalisation in an explicit substitution calculus, Rapport de Recherche No 2762, INRIA.

[Nederpelt 73] Nederpelt, R.P., *Strong normalisation in a typed lambda calculus with lambda structured types*, Ph.D. thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, 1973. Also appears in [NGV 94].

[NGV 94] Nederpelt, R.P., Geuvers, J.H. and de Vrijer, R.C., eds., *Selected Papers on Automath*, North Holland, 1994.

[Reg 92] Regnier, L., Lambda calcul et réseaux, Thèse de doctorat de l'université Paris 7, 1992.

[Reg 94] Regnier, L., Une équivalence sur les lambda termes, *Theoretical Computer Science 126*, 281-292, 1994.

[SF 92] Sabry, A., and Felleisen, M., Reasoning about programs in continuation-passing style, *Proc. 1992 ACM Conf. LISP Funct. Program.*, 288-298, 1992.

[Vid 89] Vidal, D., *Nouvelles notions de réduction en lambda calcul*, Thèse de doctorat, Université de Nancy 1, 1989.

[Whitehead and Russell 1910] Whitehead, A.N. and Russell, B., *Principia Mathematica* (Cambridge, Cambridge University Press, 1910/1913). Reprint 1960, same editor.