Explicit Extensions in (Typed) $\lambda$-calculi

Fairouz Kamareddine
Dept. of Computing & Electrical Engineering
Heriot-Watt University

10 April 2000
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.

- $(\beta) \quad (\lambda x. A)B \rightarrow_{\beta} A[x := B]$ becomes

- $(\beta) \quad (B)[x]A \rightarrow_{\beta} A[x := B]$ or $(B)[x]A \rightarrow_{\beta} [x := B]A$

- Sometimes, de Bruijn wrote: $(\beta) \quad (B)[x]A \rightarrow_{\beta} (B)[x][x := B]A$
Redexes in Item Notation

Classical Notation

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

Item Notation

\[
\begin{align*}
(a)(b)[x](c)[y][z](d)z & \rightarrow_\beta \\
(a)(c)[y][z](d)z & \rightarrow_\beta \\
(a)[z](d)z & \rightarrow_\beta (d)a
\end{align*}
\]

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 $\bar{s}$ to be a segment of partnered () and [] pairs that match like ‘‘{’ and ‘}’.

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

- \((\text{general } \beta)\) \((b)\overline{s}[v]a \rightarrow_{\beta} \overline{s}\{a[v := b]\} \quad \text{if } \overline{s} \text{ is well-balanced}\)

- Many step general \(\beta\)-reduction \(\sim_{\beta}\) is the reflexive transitive closure of \(\sim_{\beta}\).

\[
t \equiv (a)(b)[x](c)[y][z](d)z \quad \sim_{\beta}
\]

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

Lemma 1. If \(a \rightarrow_{\beta} b\) then \(a \sim_{\beta} b\). And, If \(a \sim_{\beta} b\) then \(a =_{\beta} b\).

Corollary 1. If \(a \sim_{\beta} b\) then \(a =_{\beta} b\).

Theorem 1. The general \(\beta\)-reduction is Church-Rosser. I.e. If \(a \sim_{\beta} b\) and \(a \sim_{\beta} c\), then there exists \(d\) such that \(b \sim_{\beta} d\) and \(c \sim_{\beta} d\).
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 \(\beta\)-axiom and we can contract redexes in any order.

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

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 $\beta$-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 $\lambda$-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 $\bar{s}$ be the body of a term $a$. Then the following holds:

1. Each bachelor main abstraction item in $\bar{s}$ precedes each bachelor main application item in $\bar{s}$.

2. $\bar{s}$ minus all bachelor main items equals a well-balanced segment.

3. The removal from $\bar{s}$ of all main reducible couples, leaves behind $[v_1]...[v_n](a_1)...(a_m)$, the segment consisting of all bachelor main abstraction and application items.

4. If $\bar{s} \equiv \bar{s}_1(b)\bar{s}_2[v]\bar{s}_3$ where $[v]$ and $(b)$ match, then $\bar{s}_2$ is well-balanced.
Corollary 2. For each non-empty segment $\overline{s}$, there is a unique partitioning into segments $\overline{s_0}, \overline{s_1}, \ldots, \overline{s_n}$, 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 $\overline{s_{2i}} \neq \emptyset$. \hfill $\Box$

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 \( (\lambda_x) \) or \( (\epsilon\lambda_x) \); and \([x : A]\) by \( (A\lambda_x) \).

• New items: substitution items \( (A\sigma_x) \) and typing items \( (\Gamma\tau) \).

• For example:

\[
(\beta) \quad (B\delta)(\lambda_x)A \rightarrow_{\beta} (B\delta)(\lambda_x)(B\sigma_x)A
\]
Type Theory in Item Notation

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

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

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

  \[
  \begin{align*}
  \mathcal{I}(A) & = A & \text{if } A \in \{*, \Box\} \cup V \\
  \mathcal{I}(\pi_{x:A}.B) & = (\mathcal{I}(A)\pi_{x})\mathcal{I}(B) \\
  \mathcal{I}(AB) & = (\mathcal{I}(B)\delta)\mathcal{I}(A)
  \end{align*}
  \]

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

- (\(\beta\)) \((C\delta)(B\lambda_{x})A \rightarrow_{\beta} (C\sigma_{x})A\)
Trees

Figure 3: binary tree of \((\lambda x : z. xy)u\)

Figure 4: layered tree of \((\lambda x : z. xy)u\)

\[ I((\lambda x : z. xy)u) \equiv (u\delta)(z\lambda_x)(y\delta)x \]
Compatibility

- In Classical notation:
  - \[
  \frac{A_1 \rightarrow A_2}{A_1 B \rightarrow A_2 B} \quad \frac{B_1 \rightarrow B_2}{AB_1 \rightarrow AB_2}
  \]
  - \[
  \frac{A_1 \rightarrow A_2}{\pi : A_1 : B \rightarrow \pi : A_2 : B} \quad \frac{B_1 \rightarrow B_2}{\pi : A : B_1 \rightarrow \pi : A : B_2}
  \]
- In Item notation:
  - \[
  \frac{A_1 \rightarrow A_2}{(A_1 \omega)B \rightarrow (A_2 \omega)B} \quad \frac{B_1 \rightarrow B_2}{(A \omega)B_1 \rightarrow (A \omega)B_2}
  \]
Restrictions of terms

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

- the type of $x^\circ$ in $t$ is the type of $x^\circ$ in $t \upharpoonright x^\circ$,

- the $\lambda$’s relevant to $x^\circ$ in $t$ appear also in $t \upharpoonright x^\circ$ and have the same binding relation to $x^\circ$,

- If in $t$, any substitution for $x^\circ$ is possible, then it is also possible in $t \upharpoonright x^\circ$. 

Example of term restriction

- \( t \equiv (\star \lambda_x)(x \lambda_v)(x \delta)(\star \lambda_y)((x \lambda_z)y^\circ \delta)(y \lambda_u)u. \)

- Only \((\star \lambda_x), (x \lambda_v), (x \delta), (\star \lambda_y)\) and \((x \lambda_z)\) are of importance for \( y^\circ \).
  - \( y^\circ \) is in the scope of \((\star \lambda_x), (x \lambda_v), (\star \lambda_y)\) and \((x \lambda_z)\).
  - The \( x \) is a candidate for substitution for \( y^\circ \), due to the presence of the \( \delta \lambda \)-segment \((x \delta)(\star \lambda_y)\) 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 \( y^\circ \).

- \( t \upharpoonright y^\circ \) is \((\star \lambda_x)(x \lambda_v)(x \delta)(\star \lambda_y)(x \lambda_z)\). Remove everything to the right of \( y^\circ \): \((\star \lambda_x)(x \lambda_v)(x \delta)(\star \lambda_y)((x \lambda_z)\). Remove all extra parentheses.

- If \( t \) is written \( \lambda_{x: \star} \lambda_{v:x} \cdot (\lambda_{y:*} \cdot (\lambda_{u:y} \cdot u) \lambda_{z:x} \cdot y^\circ) \cdot x \) then \( t \upharpoonright y^\circ \) is less obvious.
restriction trees

\[ t \equiv (\ast \lambda_x)((x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \]

\[ t \upharpoonright x^\circ \equiv (\ast \lambda_x)(x \lambda_u)(u \delta)(x \lambda_t)x^\circ \]

Figure 5: A term and its restriction to a variable
Definition of term restriction

Definition 1. \[ x^\circ \upharpoonright x^\circ \equiv x \text{ and } (t_1 \omega) t_2 \upharpoonright x^\circ \equiv \begin{cases} \quad t_1 \upharpoonright x^\circ & \text{if } x^\circ \text{ occurs in } t_1 \\ (t_1 \omega) (t_2 \upharpoonright x^\circ) & \text{if } x^\circ \text{ occurs in } t_2 \end{cases} \]

Let \( t \) be \((\lambda x)((x \lambda u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u\).

Then \( t \upharpoonright x^\circ \equiv (\lambda x)((x \lambda u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \)
\[ \equiv (\lambda x)((x \lambda u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)((x \lambda u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)((x \lambda u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoonright x^\circ \]
\[ \equiv (\lambda x)(x \lambda_u)((u \delta)(x \lambda_t)x^\circ \lambda_y)(u \lambda_z)y \lambda_v)u \upharpoon right
Describing normal forms in a substitution calculus

[KR 95] provided \( \lambda s \), a calculus of substitution à la de Bruijn, which remains as close as possible to the classical \( \lambda \)-calculus. The set of terms, noted \( \Lambda s \), of the \( \lambda s \)-calculus is given as follows:

\[
\Lambda s ::= \text{IN} | \Lambda s \Lambda s | \lambda \Lambda s | \Lambda s \sigma^i \Lambda s | \varphi^i_k \Lambda s \quad \text{where} \quad i \geq 1, \ k \geq 0.
\]

The set of open terms, noted \( \Lambda s_{op} \) is given as follows:

\[
\Lambda s_{op} ::= \text{V} | \text{IN} | \Lambda s_{op} \Lambda s_{op} | \lambda \Lambda s_{op} | \Lambda s_{op} \sigma \Lambda s_{op} | \varphi^i_k \Lambda s_{op} \quad \text{where} \quad i \geq 1, \ k \geq 0
\]
The $\lambda s$-calculus

\[
\begin{align*}
\sigma\text{-generation} & \quad (\lambda a) \ b & \quad \rightarrow & \quad a \sigma^1 b \\
\sigma\text{-}\lambda\text{-transition} & \quad (\lambda a) \ \sigma^i b & \quad \rightarrow & \quad \lambda (a \ \sigma^{i+1} b) \\
\sigma\text{-app-transition} & \quad (a_1 \ a_2) \ \sigma^i b & \quad \rightarrow & \quad (a_1 \ \sigma^i b) (a_2 \ \sigma^i b) \\
\sigma\text{-destruction} & \quad n \ \sigma^i b & \quad \rightarrow & \quad \begin{cases} 
\varphi^i_0 b & \text{ if } n = i \\
\varphi^i_n & \text{ if } n < i 
\end{cases} \\
\varphi\text{-}\lambda\text{-transition} & \quad \varphi^i_k (\lambda a) & \quad \rightarrow & \quad \lambda (\varphi^i_{k+1} a) \\
\varphi\text{-app-transition} & \quad \varphi^i_k (a_1 \ a_2) & \quad \rightarrow & \quad (\varphi^i_k a_1) (\varphi^i_k a_2) \\
\varphi\text{-destruction} & \quad \varphi^i_k n & \quad \rightarrow & \quad \begin{cases} 
\varphi^i_{n+i-1} & \text{ if } n > k \\
\varphi^i_n & \text{ if } n \leq k 
\end{cases}
\end{align*}
\]

We use $\lambda s$ to denote this set of rules.
The $\lambda s_e$-calculus

The $\lambda s_e$-calculus is obtained by adding the following rules to those of the $\lambda s$-calculus.

\[
\begin{align*}
\sigma-\sigma\text{-transition} & \quad (a \sigma b) \sigma^j c \quad \rightarrow \quad (a \sigma^{j+1} c) \sigma (b \sigma^{j-i+1} c) & \text{if} & \quad i \leq j \\
\sigma-\varphi\text{-transition 1} & \quad (\varphi^i_k a) \sigma^j b \quad \rightarrow \quad \varphi^{i-1}_k a & \text{if} & \quad k < j < k + i \\
\sigma-\varphi\text{-transition 2} & \quad (\varphi^i_k a) \sigma^j b \quad \rightarrow \quad \varphi^i_k (a \sigma^{j-i+1} b) & \text{if} & \quad k + i \leq j \\
\varphi-\sigma\text{-transition} & \quad \varphi^i_k (a \sigma^j b) \quad \rightarrow \quad (\varphi^i_{k+1} a) \sigma^j (\varphi^i_{k+1-j} b) & \text{if} & \quad j \leq k + 1 \\
\varphi-\varphi\text{-transition 1} & \quad \varphi^i_k (\varphi^j_l a) \quad \rightarrow \quad \varphi^j_l (\varphi^i_{k+1-j} a) & \text{if} & \quad l + j \leq k \\
\varphi-\varphi\text{-transition 2} & \quad \varphi^i_k (\varphi^j_l a) \quad \rightarrow \quad \varphi^j_{l+1-i} a & \text{if} & \quad l \leq k < l + j
\end{align*}
\]

We use $\lambda s_e$ 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:

- $a \in V \cup \text{IN}$, i.e. $a$ is a variable or a de Bruijn number.

- $a = bc$, 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 $\varphi_k^i d$ with $j \leq k$.

- $a = \varphi_k^i b$, where $b$ is an $s_e$-nf of the form $X$, or $c\sigma^j d$ with $j > k + 1$, or $\varphi_l^i c$ 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:

$$NF ::= V \mid IN \mid (NF \delta)NF \mid (\lambda)NF \mid \overline{s}V$$

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 $(\varphi^i_k)$ are called $\sigma$- and $\varphi$-items respectively. $b$ and $a$ are the bodies of these respective items.

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

$$\varphi^i_k \varphi^j_l \text{ and } k < l$$
$$b \sigma^i \sigma^j \text{ and } i < j$$

$$(\varphi^i_k \varphi^j_l \text{ and } k < j - 1)$$
$$(b \sigma^i \varphi^j_k \text{ and } j \leq k).$$
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 was the first to define a type theory ‘as such’, almost a decade after he developed a theory of functionals which is nowadays called $\lambda$-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 recently been given by Barendregt ([Bar 92]).

• Church’s $\lambda \to$ 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) \[ \langle \rangle \vdash_\beta * : \Box \]

(start rule) \[
\begin{align*}
\Gamma & \vdash_\beta A : S \\
\Gamma.\lambda x: A & \vdash_\beta x : A \quad x \not\in \Gamma
\end{align*}
\]

(weakening rule) \[
\begin{align*}
\Gamma & \vdash_\beta A : S \\
\Gamma & \vdash_\beta D : E
\end{align*}
\]

(application rule) \[
\begin{align*}
\Gamma & \vdash_\beta F : \Pi x: A. B \\
\Gamma & \vdash_\beta a : A
\end{align*}
\]

\[
\Gamma \vdash_\beta Fa : B[x := a]
\]

(abstraction rule) \[
\begin{align*}
\Gamma.\lambda x: A & \vdash_\beta b : B \\
\Gamma & \vdash_\beta \Pi x: A. B : S
\end{align*}
\]

\[
\Gamma \vdash_\beta \lambda x: A. b : \Pi x: A. B
\]

(conversion rule) \[
\begin{align*}
\Gamma & \vdash_\beta A : B \\
\Gamma & \vdash_\beta B' : S
\end{align*}
\]

\[
B =_\beta B' \\
\Gamma \vdash_\beta A : B'
\]

(formation rule) \[
\begin{align*}
\Gamma & \vdash_\beta A : S_1 \\
\Gamma & \vdash_\beta \Pi x: A. B : S_2
\end{align*}
\]

\[
\text{if } (S_1, S_2) \text{ is a rule}
\]
<table>
<thead>
<tr>
<th>System</th>
<th>Allowed ((S_1, S_2)) rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda\rightarrow)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda 2)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda P)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda P2)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda \omega)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda \omega)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda P \omega)</td>
<td>(*, *)</td>
</tr>
<tr>
<td>(\lambda P \omega = \lambda C)</td>
<td>(*, *)</td>
</tr>
</tbody>
</table>
Figure 6: The Cube
Example derivation

Take $\Gamma \equiv \lambda_{\beta:*}.\lambda_{y:\beta}$. In $\lambda_2$, using the rules $(\cdot,\cdot)$ and $(\square,\cdot)$ we have:

\[
\begin{align*}
\Gamma &\vdash_{\lambda_2} y : \beta : * : \square \\
\Gamma &\vdash_{\lambda_2} \lambda_{\alpha:*} \vdash_{\lambda_2} \alpha : * \quad \text{(start)} \\
\Gamma &\vdash_{\lambda_2} \lambda_{\alpha:*}.\lambda_{x:\alpha} \vdash_{\lambda_2} x : \alpha : * \quad \text{(start resp weakening)} \\
\Gamma &\vdash_{\lambda_2} \lambda_{\alpha:*} \vdash_{\lambda_2} \Pi_{x:\alpha}.\alpha : * \quad \text{(formation rule $(\cdot,\cdot)$)} \\
\Gamma &\vdash_{\lambda_2} \lambda_{\alpha:*}.\lambda_{x:\alpha}.x : \Pi_{x:\alpha}.\alpha \quad \text{(abstraction)} \\
\Gamma &\vdash_{\lambda_2} \Pi_{\alpha:*}.\Pi_{x:\alpha}.\alpha : * \quad \text{(formation rule $(\square,\cdot)$)} \\
\Gamma &\vdash_{\lambda_2} \lambda_{\alpha:*}.\lambda_{x:\alpha}.x : \Pi_{\alpha:*}.\Pi_{x:\alpha}.\alpha \quad \text{(abstraction)} \\
\Gamma &\vdash_{\lambda_2} (\lambda_{\alpha:*}.\lambda_{x:\alpha})\beta : \Pi_{x:\beta}.\beta \quad \text{(application, we already knew $\Gamma \vdash_{\lambda_2} \beta : *$)} \\
\Gamma &\vdash_{\lambda_2} (\lambda_{\alpha:*}.\lambda_{x:\alpha}.x)\beta y : \beta \quad \text{(application, we already knew $\Gamma \vdash_{\lambda_2} y : \beta$)}
\end{align*}
\]

It is not possible to derive this judgement in $\lambda_\rightarrow$ as $(\square,\cdot)$ is needed.
The system $\lambda \rightarrow$
(axiom) \[ \langle \rangle \vdash_{\beta} * : \Box \]

(start rule) \[ \frac{\Gamma \vdash_{\beta} A : S}{\Gamma.\lambda_{x:A} \vdash_{\beta} x : A} \quad x \not\in \Gamma \]

(weakening rule) \[ \frac{\Gamma \vdash_{\beta} A : S \quad \Gamma \vdash_{\beta} D : E}{\Gamma.\lambda_{x:A} \vdash_{\beta} D : E} \quad x \not\in \Gamma \]

(application rule) \[ \frac{\Gamma \vdash_{\beta} F : A \to B \quad \Gamma \vdash_{\beta} a : A}{\Gamma \vdash_{\beta} Fa : B} \]

(abstraction rule) \[ \frac{\Gamma.\lambda_{x:A} \vdash_{\beta} b : B \quad \Gamma \vdash_{\beta} A \to B : S}{\Gamma \vdash_{\beta} \lambda_{x:A}.b : A \to B} \]

(conversion rule) \[ \frac{\Gamma \vdash_{\beta} A : B \quad \Gamma \vdash_{\beta} B' : S \quad B =_{\beta} B'}{\Gamma \vdash_{\beta} A : B'} \]

(formation rule) \[ \frac{\Gamma \vdash_{\beta} A : \ast \quad \Gamma.\lambda_{x:A} \vdash_{\beta} B : \ast}{\Gamma \vdash_{\beta} \Pi_{x:A}.B : \ast} \]
The system $\lambda \rightarrow$ revised

(start rule) \[ \Gamma \vdash_{\beta} A : S \quad \Gamma, \lambda_{x:A} \vdash_{\beta} x : A \quad x \notin \Gamma \]

(weakening rule) \[ \Gamma \vdash_{\beta} A : S \quad \Gamma \vdash_{\beta} D : E \quad \Gamma, \lambda_{x:A} \vdash_{\beta} D : E \quad x \notin \Gamma \]

(application rule) \[ \Gamma \vdash_{\beta} F : A \rightarrow B \quad \Gamma \vdash_{\beta} a : A \quad \Gamma \vdash_{\beta} Fa : B \]

(abstraction rule) \[ \Gamma \vdash_{\beta} b : B \quad \Gamma, \lambda_{x:A} \vdash_{\beta} b : A \rightarrow B \]
Desirable Properties: See [Bar 92]

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

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

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 $\rightarrow_\beta$)

If $\Gamma \vdash_\beta A : B$ and $A \rightarrow_\beta A'$ then $\Gamma \vdash_\beta A' : B$

Theorem 4. (Strong Normalisation with respect to $\vdash_\beta$ and $\rightarrow_\beta$)

For all $\vdash_\beta$-legal terms $M$, we have $SN_{\rightarrow_\beta}(M)$. I.e. $M$ is strongly normalising with respect to $\rightarrow_\beta$. 

EEF Foundations school in Deductions and Theorem Proving, Heriot-Watt univ, April 6-16 2000
\(\Pi\)-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: \((\lambda_x: A. b) C \rightarrow_\beta b[x := C]\), as well as \((\Pi_x: A. B) C \rightarrow_\beta B[x := C]\).

- This strategy of permitting \(\Pi\)-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).
  
  If \(f : \Pi_x: A. B\) and \(a : A\), then \(fa : B[x := a]\) \hspace{1cm} (1)

  If \(f : \Pi_x: A. B\) and \(a : A\), then \(fa : (\Pi_x: A. B)a\). \hspace{1cm} (2)
• With \( \Pi \)-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:

\[
M : N \Rightarrow \lambda y : PM : \Pi y : PN.
\]

\[
\begin{align*}
A : *, b : A, a : A & \vdash a : A & \text{(start)} \\
A : *, b : A & \vdash (\lambda a : A.a) : (\Pi a : A.A) & \text{(abstraction)} \\
A : *, b : A & \vdash (\lambda a : A.a)b : (\Pi a : A.A)b & \text{(application)} \\
A : *, b : A & \vdash (\lambda a : A.a)b : A & \text{(conversion)}
\end{align*}
\]

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

- In a compiler, $\Pi$-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 \( \Pi \)-reduction: See [KN 96a]

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

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

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

\begin{align*}
\text{(new application rule)} \quad & \frac{\Gamma \vdash_{\beta \Pi} F : \Pi_{x:A}.B \quad \Gamma \vdash_{\beta \Pi} \ a : A}{\Gamma \vdash_{\beta \Pi} Fa : (\Pi_{x:A}.B)a} \\
\text{(new conversion rule)} \quad & \frac{\Gamma \vdash_{\beta \Pi} A : B \quad \Gamma \vdash_{\beta \Pi} B' : S \quad B =_{\beta \Pi} B'}{\Gamma \vdash_{\beta \Pi} A : B'}
\end{align*}
Barendregt Cube with Π-reduction
(axiom) \(\iff \Gamma \vdash_{\beta \Pi} A : S\)

(start rule) \(\Gamma \vdash_{\beta \Pi} A : S\)

(weakening rule) \(\Gamma \vdash_{\beta \Pi} A : S\)

(new application rule) \(\Gamma \vdash_{\beta \Pi} F : \Pi_{x:A} B\)

(abstraction rule) \(\Gamma \vdash_{\beta \Pi} b : B\)

(new conversion rule) \(\Gamma \vdash_{\beta \Pi} A : B\)

(formation rule) \(\Gamma \vdash_{\beta \Pi} A : S_1\)
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 $\beta$ to to $\beta\Pi$ everywhere.
Correctness of types fails for $\Pi$-reduction even in $\lambda\to$

**Lemma 5.** For any $A, B, C, S$ we have $\Gamma \vdash_{\beta\Pi} (\Pi_{x:A}.B)C : S$.

**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, $\Gamma.\lambda_{x:A} \vdash_{\beta\Pi} B : S' \land S' =_{\beta\Pi} \Pi_{x:A'}.B'$. Absurd. \qed

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 $\Pi$-redex then ($B \equiv \square$ 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 $\Pi$-redex. \hfill \square

Failure of correctness of types implies failure of Subject Reduction even in $\lambda\rightarrow$:

- In $\lambda\rightarrow$, we have: $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} x : (\Pi_{y:z}.z)x$.

- 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 $\lambda\rightarrow$, we have: $\lambda_{z:*}.\lambda_{x:z} \vdash_{\beta\Pi} (\lambda_{y:z.y})x : (\Pi_{y:z}.z)x$. 

---

EEF Foundations school in Deductions and Theorem Proving, Heriot-Watt univ, April 6-16 2000
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 $\rightarrow_{\beta \Pi}$)

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

2. If $\Gamma \vdash_{\beta \Pi} A : B$ and $A \rightarrow_{\beta \Pi} A'$ 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 $\tau$) 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 $A \equiv \lambda x:\ast\ast(\lambda y:\ast\ast y)x$ is $\tau(\langle\rangle, A) \equiv \Pi x:\ast(\Pi y:\ast x)$ which $\beta\Pi$ to $\Pi y:\ast x$, the type traditionally given to $A$.

- The conversion rule is no longer needed as a separate rule in the definition of
\( \vdash \text{It is accommodated in our application rule:} \)

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash AB} \quad \text{if } \tau(\Gamma, A) = \beta\Pi \Pi_x:C.D \text{ and } \tau(\Gamma, B) = \beta\Pi C
\]

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

- Higher degrees: If we use \( \lambda^1 \) for \( \Pi \) and \( \lambda^2 \) for \( \lambda \) then we can aim for a possible generalization. In fact, we can extend our system by incorporating more different \( \lambda \)'s. For example, with an infinity of \( \lambda \)'s, viz. \( \lambda^0, \lambda^1, \lambda^2, \lambda^3 \ldots \), 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:

\[
\tau(\Gamma, \lambda^{i+1}_{x:A}.B) \equiv \lambda^i_{x:A}.\tau(\Gamma, \lambda_x:A, B), \text{ for } i = 0, 1, 2, \ldots \text{ where } \lambda^0_{x:A}.B \equiv B
\]
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 $\tau(\Gamma, A) = B$.

\[
\begin{align*}
\tau(\Gamma, *) &\equiv \square \\
\tau(\Gamma, x) &\equiv A \text{ if } (A\lambda_x) \in \Gamma \\
\tau(\Gamma, (a\delta)F) &\equiv (a\delta)\tau(\Gamma, F) \\
\tau(\Gamma, (A\lambda_x)B) &\equiv (A\Pi_x)\tau(\Gamma(A\lambda_x), B) \quad \text{if } x \not\in \text{dom}(\Gamma) \\
\tau(\Gamma, (A\Pi_x)B) &\equiv \tau(\Gamma(A\lambda_x), B) \quad \text{if } x \not\in \text{dom}(\Gamma)
\end{align*}
\]
• In usual type theory:

- the type of \((\ast \lambda_x)(x \lambda_y)y\) is \((\ast \Pi_x)(x \Pi_y)x\) and
- the type of \((\ast \Pi_x)(x \Pi_y)x\) is \(*\).

• With our \(\tau\), we get the same result:

- \(\tau(\langle\rangle, (\ast \lambda_x)(x \lambda_y)y) \equiv (\ast \Pi_x)\tau((\ast \lambda_x), (x \lambda_y)y) \equiv (\ast \Pi_x)(x \Pi_y)\tau((\ast \lambda_x)(x \lambda_y), y) \equiv (\ast \Pi_x)(x \Pi_y)x\) and
- \(\tau(\langle\rangle, (\ast \Pi_x)(x \Pi_y)x) \equiv \tau((\ast \lambda_x), (x \Pi_y)x) \equiv \tau((\ast \lambda_x)(x \lambda_y), x) \equiv *\)
Let $\Gamma_0 \equiv \langle \rangle$, $\Gamma_1 \equiv (*\lambda_z)$, $\Gamma_2 \equiv (*\lambda_z)(*\lambda_y)$, $\Gamma_3 \equiv \Gamma_2(*\lambda_x)$. We want to find the canonical type of $(*\Pi_z)(B\delta)(*\lambda_y)(y\delta)(*\lambda_x)x$ in $\Gamma_0$.

\[
\begin{array}{cccccc}
\text{(}\Gamma_0\tau) & (*\Pi_z) & (B\delta) & (*\lambda_y) & (y\delta) & (*\lambda_x) & x
\end{array}
\]

\[
\begin{array}{cccccc}
\text{(}\Gamma_1\tau) & (B\delta) & (*\lambda_y) & (y\delta) & (*\lambda_x) & x
\end{array}
\]

\[
\begin{array}{cccccc}
(B\delta) & (\Gamma_1\tau) & (*\lambda_y) & (y\delta) & (*\lambda_x) & x
\end{array}
\]

\[
\begin{array}{cccccc}
(B\delta) & (*\Pi_y) & (\Gamma_2\tau) & (y\delta) & (*\lambda_x) & x
\end{array}
\]

\[
\begin{array}{cccccc}
(B\delta) & (*\Pi_y) & (y\delta) & (\Gamma_2\tau) & (*\lambda_x) & x
\end{array}
\]

\[
\begin{array}{cccccc}
(B\delta) & (*\Pi_y) & (y\delta) & (\Gamma_2\tau) & (*\Pi_x) & (\Gamma_3\tau) & x
\end{array}
\]
**New Typability**

(\(\vdash\)-axiom) \[<\!> \vdash *\]

(\(\vdash\)-start rule) \[
\frac{\Gamma \vdash A}{\Gamma(A\lambda_x) \vdash x} \text{ if vc}
\]

(\(\vdash\)-weakening rule) \[
\frac{\Gamma \vdash A \quad \Gamma \vdash D}{\Gamma(A\lambda_x) \vdash D} \text{ if vc}
\]

(\(\vdash\)-application rule) \[
\frac{\Gamma \vdash F \quad \Gamma \vdash a}{\Gamma \vdash (a\delta)F} \text{ if ap}
\]

(\(\vdash\)-abstraction rule) \[
\frac{\Gamma(A\lambda_x) \vdash b \quad \Gamma \vdash (A\Pi_x)B}{\Gamma \vdash (A\lambda_x)b} \text{ if ab}
\]

(\(\vdash\)-formation) \[
\frac{\Gamma \vdash A \quad \Gamma(A\lambda_x) \vdash B}{\Gamma \vdash (A\Pi_x)B} \text{ if fc}
\]
• vc (variable condition): $x \not\in \Gamma$ and $\tau(\Gamma, A) \rightarrow_{\beta_\Pi} S$ 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 $\tau(\Gamma, (A\Pi_x)B) \rightarrow_{\beta_\Pi} S$ for some $S$.

• fc (formation condition): $\tau(\Gamma, A) \rightarrow_{\beta_\Pi} S_1$ and $\tau(\Gamma, A\lambda_x), B) \rightarrow_{\beta_\Pi} S_2$ 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 $\downarrow \tau(\Gamma, A)$ and $\Gamma \vdash_\beta A : \tau(\Gamma, A)$

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

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

**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

\[ C(CR, SN, SR) \]

\[ C_{\sim \beta}(CR, SN) \quad C_{def}(CR, SN, SR) \]

\[ C_{\sim \beta_{def}}(CR, SN, SR) \]

Figure 7: Properties of the Cube with generalised reduction
References


[KR 95] Kamareddine, F., and Rios, A., $\lambda$-calculus à la de Bruijn & explicit substitution, *Lecture Notes in Computer Science* 982, 7th international


[KW 94] Kfoury, A.J. and Wells, J.B., A direct algorithm for type inference in


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

[BLR 95] Benaissa, Briaud, Lescanne, Rouyer-Degli, $\lambda v$, 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.


