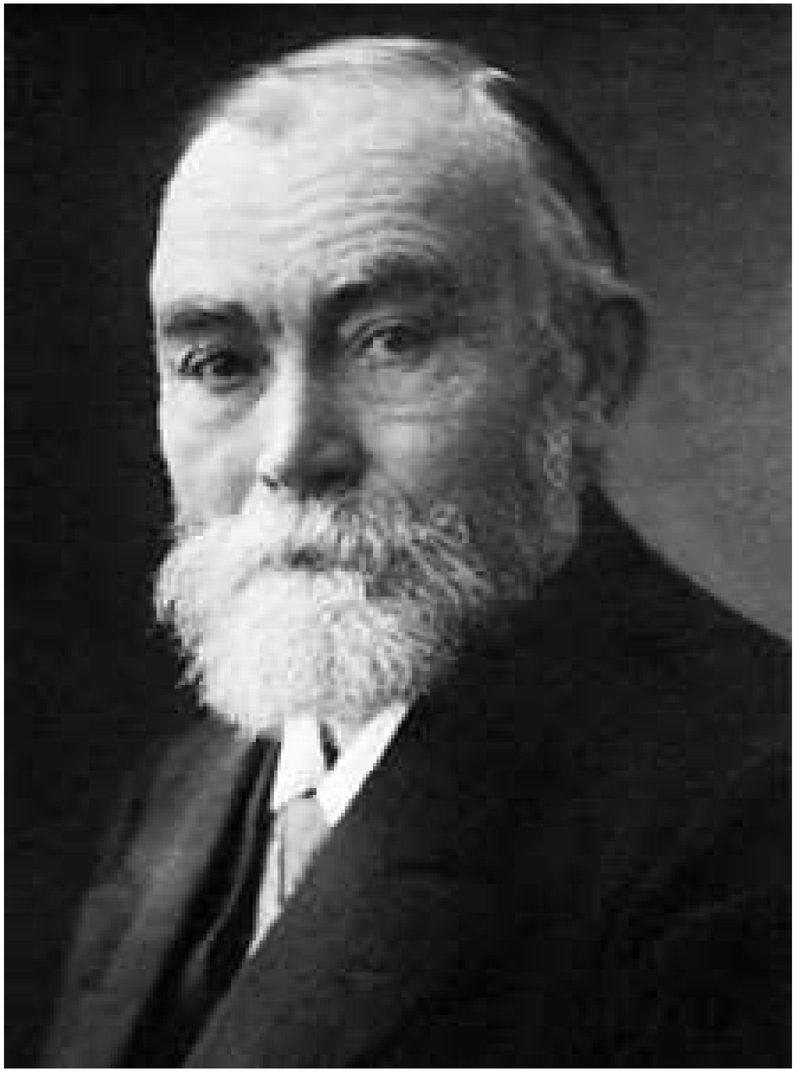
Automath type inclusion in Barendregt's Cube

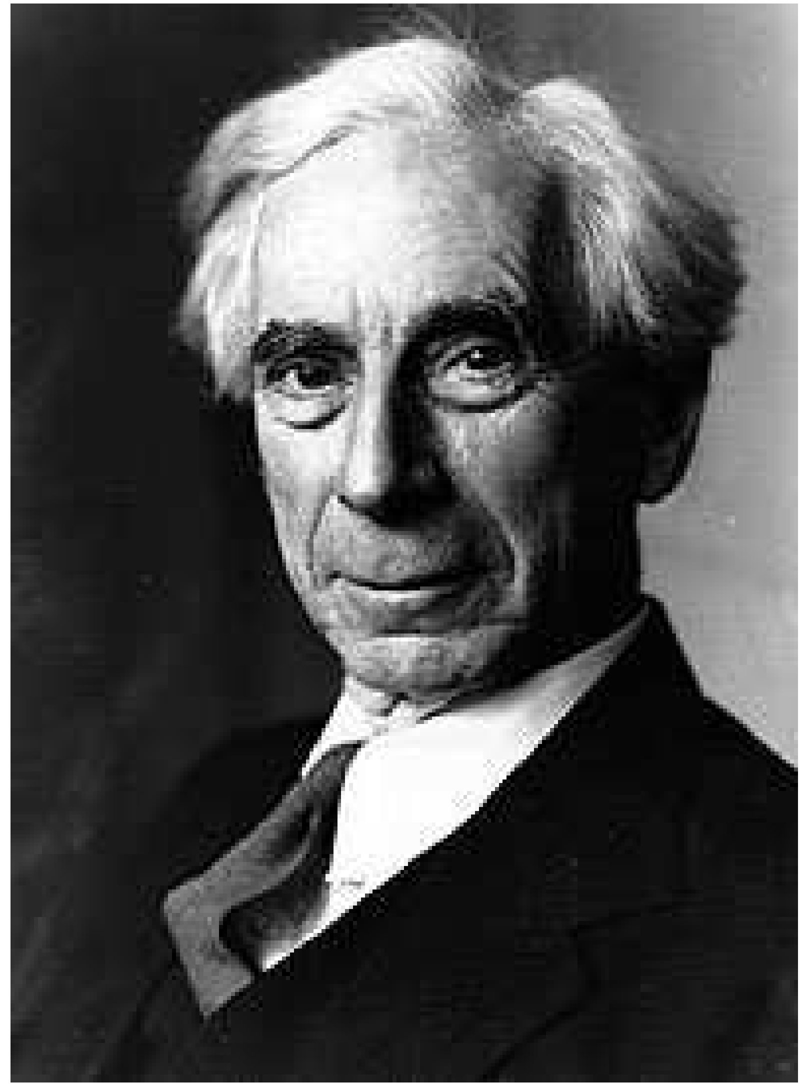
Fairouz Kamareddine & Joe Wells (Heriot-Watt University, Edinburgh, UK)
Daniel ventura (Universidade Federal de Goiás, Goiânia GO, Brazil)

July 2015

Development of Types and Functions

- *General definition of function* [Frege, 1879] is key to his *formalisation of logic*.
- *Self-application of functions* was the heart of *Russell's paradox* [Russell, 1902].
- To *avoid paradox* Russell controled function application via *type theory*.
- [Russell, 1903] gives the first type theory: the *Ramified Type Theory* (RTT).
- RTT is used in Principia Mathematica [Whitehead and Russell, 1910^1, 1927^2].
- *Simple theory of types* (STT): [Ramsey, 1926], [Hilbert and Ackermann, 1928].
- The hierarchies of types/orders of RTT and STT are *unsatisfactory*.





Development of Types and Functions continued

- In the 30s Church gave λ-calculus where all functions are 1st-class citizens.
- Frege's functions $\neq$ Principia's functions $\neq$ *λ-calculus* functions.
- In 1940 Church gave *simply typed λ-calculus* $\lambda\!\rightarrow\ = \lambda$-calculus $+$ STT.
- Not all functions need to be *fully abstracted* as in the λ-calculus.
- For some functions, values are enough. E.g., we speak of $\sin(x)$ not of $\sin$.
- Without function definitions/abbreviations, e.g. $x = A$ and $x = A$ in B, mathematics would be infeasible.
- Developments of ordinary mathematical theory in AUTOMATH [Benthem Jutting, 1977] revealed that this combined definition and function value mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

- *Non-first-class functions* allow us to stay at a lower order (keeping decidability, typability, etc.) without losing the flexibility of the higher-order aspects.

- Frege and Russell's notions of low level functions (which was lost in Church's notion of function) allow us not only keep feasibility, but also solve important problems like correctness of typability and subject reduction (safety).

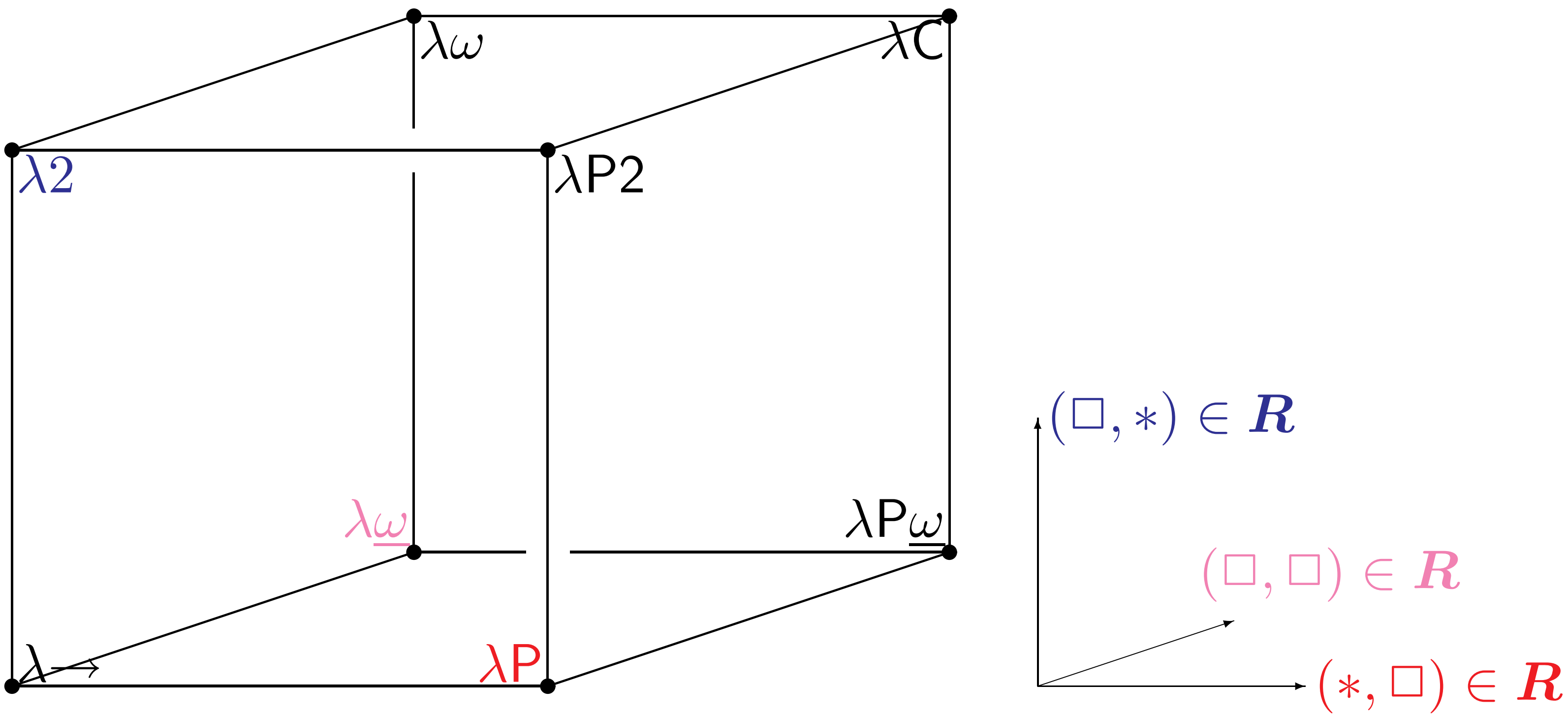
- Non-first-class functions allow placing the type systems of modern theorem provers/programming languages like ML, LF and Automath more accurately in the modern hierarchy of types.

The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \square \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- Formation rule (Π): $$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2}$$ if $(s_1, s_2) \in \boldsymbol{R}$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda{\to}$	$(*,*)$				λ^τ	[Church, 1940; B
$\lambda 2$	$(*,*)$	$(\square,*)$			F	[Girard, 1972; Re
λP	$(*,*)$		$(*,\square)$		AUT-QE, LF	[Bruijn, 1968; Ha
$\lambda\underline{\omega}$	$(*,*)$			$(\square,\square)$	POLYREC	[Renardel de Lava
λP2	$(*,*)$	$(\square,*)$	$(*,\square)$			[Longo and Mogg
$\lambda\omega$	$(*,*)$	$(\square,*)$		$(\square,\square)$	Fω	[Girard, 1972]
λP$\underline{\omega}$	$(*,*)$		$(*,\square)$	$(\square,\square)$		
λC	$(*,*)$	$(\square,*)$	$(*,\square)$	$(\square,\square)$	CC	[Coquand and Hu

The Barendregt Cube



The β-cube: $\to_\beta$ + conv$_\beta$ + app$_\Pi$

(axiom)	$\langle\rangle \vdash * : \square$
(start)	$\dfrac{\Gamma \vdash A : s \quad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}A \vdash x : A}$
(weak)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}C \vdash A : B}$
(Π)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$
(λ)	$\dfrac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$
(conv$_\beta$)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$
(app$_\Pi$)	$\dfrac{\Gamma \vdash F : \Pi_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$

Typing Polymorphic identity needs $(\Box, *)$

- $\dfrac{y : * \vdash y : * \qquad y : *, x{:}y \vdash y : *}{y : * \vdash \Pi x{:}y.y : *}$ by (Π) $(*, *)$

- $\dfrac{y : *, x : y \vdash x : y \qquad y : * \vdash \Pi x{:}y.y : *}{y : * \vdash \lambda x : y.x : \Pi x{:}y.y}$ by (λ)

- $\dfrac{\vdash * : \Box \qquad y : * \vdash \Pi x{:}y.y : *}{\vdash \Pi y : *.\Pi x{:}y.y : *}$ by (Π) $(\Box, *)$

- $\dfrac{y : * \vdash \lambda x : y.x : \Pi x{:}y.y \qquad \vdash \Pi y : *.\Pi x{:}y.y : *}{\vdash \lambda y : *.\lambda x : y.x : \Pi y : *.\Pi x{:}y.y}$ by (λ)

6 desirable properties of a type system with reduction r

- *Types are correct (TC)*
  If $\Gamma \vdash A : B$ then $B \equiv \Box$ or $\Gamma \vdash B : s$ for $s \in \{*, \Box\}$.

- *Subject reduction (SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_r A'$ then $\Gamma \vdash A' : B$.

- *Preservation of types (PT)* If $\Gamma \vdash A : B$ and $B \twoheadrightarrow_r B'$ then $\Gamma \vdash A : B'$.

- *Strong Normalisation (SN)* If $\Gamma \vdash A : B$ then $\mathrm{SN}_{\rightarrow_r}(A)$ and $\mathrm{SN}_{\rightarrow_r}(B)$.

- *Subterms are typable (STT)* If A is $\vdash$-legal and if C is a sub-term of A then C is $\vdash$-legal.

- *Unicity of types*
  - *(UT1)* If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $\Gamma \vdash A_1 =_r A_2$, then $\Gamma \vdash B_1 =_r B_2$.
  - *(UT2)* If $\Gamma \vdash B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s$.

The evolution of functions with Frege, Russell and Church

- Historically, functions have long been treated as a kind of meta-objects.
- Function *values* were the important part, not abstract functions.
- In the *low level/operational approach* there are only function values.
- The sine-function, is always expressed with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- In many mathematics courses, one calls $f(x)$—and not f—the function.
- Frege, Russell and Church wrote $x \mapsto x + 3$ resp. as $x + 3$, $\hat{x} + 3$ and $\lambda x.x + 3$.
- Principia's *functions are based on Frege's Abstraction Principles* but can be first-class citizens. Frege used courses-of-values to speak about functions.
- Church made every function a first-class citizen. This is rigid and does not represent the development of logic in 20th century.

From the point of vue of ML

- The language ML is not based on all of system F (2nd order λ-calculus).
- This was not possible since it was not known then whether type checking and type finding are decidable.
- ML is based on a fragment of system F for which it was known that type checking and type finding are decidable.
- 23 years later after the design of ML, Joe Wells showed that type checking and type finding in system F are undecidable.
- ML has polymorphism but not all the polymorphic power of system F.

- The question is, what system of functions and types does ML use?
- *A clean answer can be given when we re-incorporate the low-level function notion used by Frege and Russell (and de Bruijn) and dismissed by Church.*
- ML treats let val id $=$ (fn $x \Rightarrow x$) in (id id) end as this Cube term $(\lambda \mathsf{id}{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ \mathsf{id}(\beta \to \beta)(\mathsf{id}\,\beta))(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$
- To type this in the Cube, the $(\Box, *)$ rule is needed (i.e., $\lambda 2$).

- ML's typing rules forbid this expression:
  let val id $=$ (fn $x \Rightarrow x$) in (fn $y \Rightarrow y\,y$)(id id) end
  Its equivalent Cube term is this well-formed typable term of $\lambda 2$:
  $(\lambda \mathrm{id} : (\Pi\alpha{:}*.\ \alpha \to \alpha).$
  $\quad(\lambda y{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ y(\beta \to \beta)(y\,\beta))$
  $\quad(\lambda\alpha{:}*.\ \mathrm{id}(\alpha \to \alpha)(\mathrm{id}\,\alpha)))$
  $(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$
- Therefore, ML should not have the full Π-formation rule $(\Box, *)$.
- ML has limited access to the rule $(\Box, *)$.
- ML's type system is none of those of the eight systems of the Cube.

LF

- LF [Harper et al., 1987] is often described as λP of the Barendregt Cube. However, Use of Π-formation rule $(*, \Box)$ is restricted in LF [Geuvers, 1993].
- We only need a type $\Pi x{:}A.B : \Box$ when PAT is applied during construction of the type $\Pi\alpha{:}\texttt{prop}.*$ of the operator Prf where for a proposition Σ, Prf(Σ) is the type of proofs of Σ.

$$\frac{\texttt{prop}{:}* \vdash \texttt{prop}{:}\ * \qquad \texttt{prop}{:}*, \alpha{:}\texttt{prop} \vdash *{:}\Box}{\texttt{prop}{:}* \vdash \Pi\alpha{:}\texttt{prop}.* : \Box}.$$

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used. But, Prf is only used when applied to $\Sigma{:}\texttt{prop}$. We never use Prf on its own.
- This use is in fact based on a parametric constant rather than on Π-formation.
- Hence, the practical use of LF would not be restricted if we present Prf in a parametric form, and use $(*, \Box)$ as a parameter instead of a Π-formation rule.

Parameters: What and Why

- We speak about *functions with parameters* when referring to functions with variable values in the *low-level* approach. The x in $f(x)$ is a parameter.

- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. first-order with parameters versus second-order without [Laan and Franssen, 2001].

- Desirable properties of the lower order theory (decidability, easiness of calculations, typability) can be maintained, without losing the flexibility of the higher-order aspects.

- This low-level approach is still worthwhile for many exact disciplines. In fact, both in logic and in computer science it has certainly not been wiped out, and for good reasons.

Extending the Cube with parametric constants, see K. etal'01

- We add parametric constants of the form $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types and $c \in \mathcal{C}$.

- $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.

- $\mathbf{R}$ allows several kinds of Π-constructs. We also use a set $\boldsymbol{P}$ of (s_1, s_2) where $s_1, s_2 \in \{*, \Box\}$ to allow several kinds of parametric constants.

- $(s_1, s_2) \in \boldsymbol{P}$ means that we allow parametric constants $c(b_1, \ldots, b_n) : A$ where $b_1, \ldots, b_n$ have types $B_1, \ldots, B_n$ of sort s_1, and A is of type s_2.

- If both $(*, s_2) \in \boldsymbol{P}$ and $(\Box, s_2) \in \boldsymbol{P}$ then combinations of parameters allowed. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\Box$.

The Cube with parametric constants

- Let $(*,*) \subseteq \mathbf{R}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.

- $\lambda\mathbf{R}\boldsymbol{P} = \lambda\mathbf{R}$ and the two rules **($\vec{\mathsf{C}}$-weak)** and **($\vec{\mathsf{C}}$-app)**:

$$\frac{\Gamma \vdash b : B \qquad \Gamma, \Delta_i \vdash B_i : s_i \qquad \Gamma, \Delta \vdash A : s}{\Gamma, c(\Delta) : A \vdash b : B} \; (s_i, s) \in \boldsymbol{P}, c \text{ is } \Gamma\text{-fresh}$$

$$\frac{\begin{array}{lcll} \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n) \\ \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & A : s & (\text{if } n = 0) \end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

$\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$.
$\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$

Properties of the Refined Cube

- (Correctness of types) If $\Gamma \vdash A : B$ then $(B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort $S)$.
- (Subject Reduction SR) If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$
- (Strong Normalisation) For all $\vdash$-legal terms M, we have $\mathsf{SN}_{\twoheadrightarrow_\beta}(M)$.
- Other properties such as Uniqueness of types and typability of subterms hold.
- $\lambda\mathbf{R}\boldsymbol{P}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.
- Let $\lambda\mathbf{R}\boldsymbol{P}$ parametrically conservative (i.e., $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$).
  - The parameter-free system $\lambda\mathbf{R}$ is at least as powerful as $\lambda\mathbf{R}\boldsymbol{P}$.
  - If $\Gamma \vdash_{\mathbf{R}\boldsymbol{P}} a : A$ then $\{\Gamma\} \vdash_{\boldsymbol{R}} \{a\} : \{A\}$.

Example

- $\boldsymbol{R} = \{(*,*),(*,\Box)\}$

  $\boldsymbol{P}_1 = \emptyset \qquad \boldsymbol{P}_2 = \{(*,*)\} \qquad \boldsymbol{P}_3 = \{(*,\Box)\} \qquad \boldsymbol{P}_4 = \{(*,*),(*,\Box)\}$

  All $\lambda\boldsymbol{R}\boldsymbol{P}_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

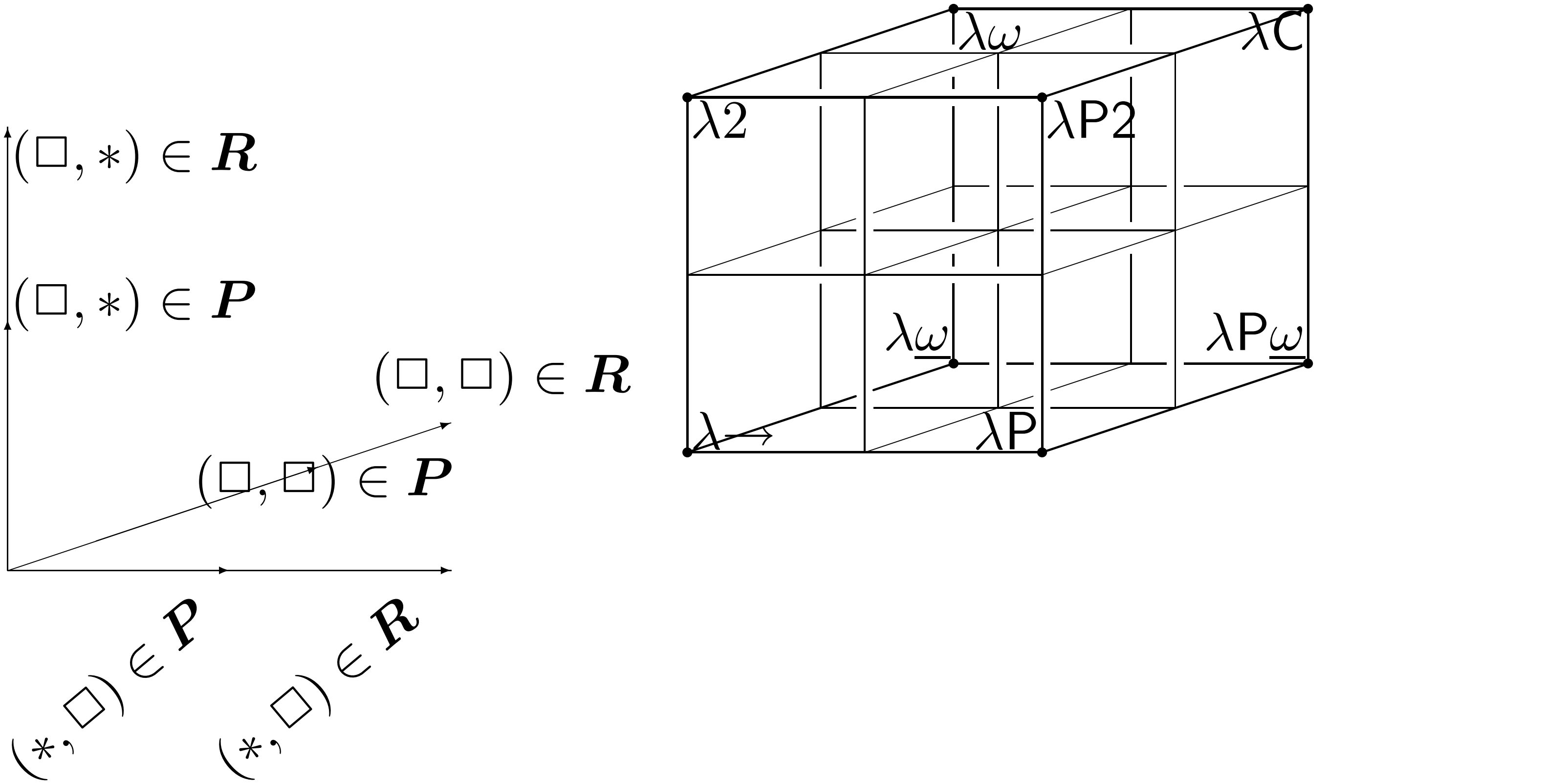
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*),(*,\Box)\}$.

  $\lambda\!\to \; < \; \lambda\boldsymbol{R}_5\boldsymbol{P}_5 \; < \; \lambda$P: we can to talk about predicates:

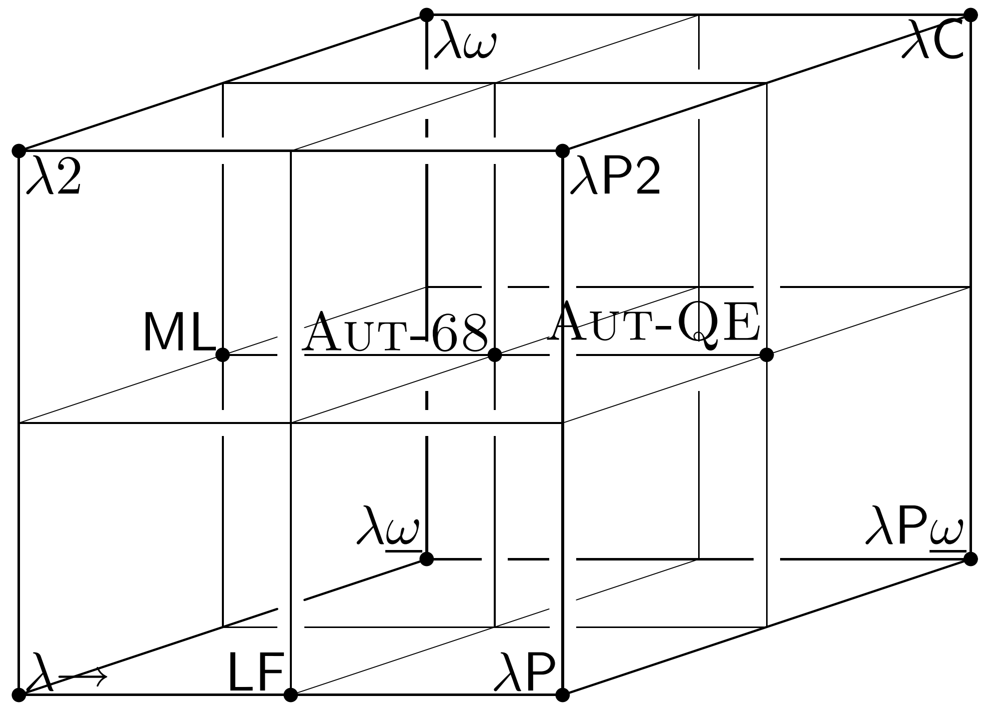
$$\begin{array}{rcl} \alpha & : & *, \\ \mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\ \mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \\ \mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & : & \mathtt{eq}(\mathtt{y},\mathtt{x}), \\ \mathtt{trans}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{z}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y}), \mathtt{q}{:}\mathtt{eq}(\mathtt{y},\mathtt{z})) & : & \mathtt{eq}(\mathtt{x},\mathtt{z}) \end{array} .$$

  eq not possible in $\lambda\!\to$.

The refined Barendregt Cube



LF, ML, AUT-68, and AUT-QE in the refined Cube



Logicians versus mathematicians and induction over numbers

- Logician uses `ind: Ind` as proof term for an application of the induction axiom. The type `Ind` can only be described in $\lambda\boldsymbol{R}$ where $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}{\to}*).p0{\to}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm){\to}\Pi n{:}\mathbb{N}.pn \qquad (1)$$

- Mathematician uses `ind` only with $P : \mathbb{N}{\to}*$, $Q : P0$ and $R : (\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.Pn{\to}Snm{\to}Pm)$ to form a term $(\texttt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.

- The use of the induction axiom by the mathematician is better described by the parametric scheme (p, q and r are the *parameters* of the scheme):

$$\texttt{ind}(p{:}\mathbb{N}{\to}*, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm)) : \Pi n{:}\mathbb{N}.pn \qquad (2)$$

- The logician's type `Ind` is not needed by the mathematician and the types that occur in 2 can all be constructed in $\lambda\boldsymbol{R}$ with $\boldsymbol{R} = \{(*,*)(*,\Box)\}$.

Logicians versus mathematicians and induction over numbers

- Mathematician: only *applies* the induction axiom and doesn't need to know the proof-theoretical backgrounds.
- A logician develops the induction axiom (or studies its properties).
- $(\Box, *)$ is not needed by the mathematician. It is needed in logician's approach in order to form the Π-abstraction $\Pi p{:}(\mathbb{N} \to *). \cdots)$.
- Consequently, the type system that is used to describe the mathematician's use of the induction axiom can be weaker than the one for the logician.
- Nevertheless, the parameter mechanism gives the mathematician limited (but for his purposes sufficient) access to the induction scheme.

Common features of modern types and functions

- Write $A \to A$ as $\Pi_{y:A}.A$ when y not free in A.

- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $\Pi_{y:A}.A$, *i.e.* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$

- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \to A)\mathbb{N} = (A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.

- More clearly

Term	$\lambda_{y:A}.y$	$\lambda_{A:*}.\lambda_{y:A}.y$	$(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$
Type	$\Pi_{y:A}.A$	$\Pi_{A:*}.\Pi_{y:A}.A$	$(\Pi_{A:*}.\Pi_{y:A}.A)\mathbb{N}$
shorthand	$A \to A$	$\Pi_{A:*}.A \to A$	$(\Pi_{A:*}.A \to A)\mathbb{N}$

- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$ $\quad$ $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$

The π-cube: $R_\pi = R_\beta \setminus$ (conv$_\beta$) $\cup$ (conv$_{\beta\Pi}$), $\to_{\beta\Pi}$

- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$
- $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$)

(conv$_{\beta\Pi}$) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_{\beta\Pi} B'}{\Gamma \vdash A : B'}$$

Lemma: $\Gamma \vdash_\beta A : B$ iff $\Gamma \vdash_\pi A : B$

Lemma: The β-cube and the π-cube satisfy the six properties that are desirable for type systems.

The π_i-cube: $R_{\pi_i} = R_\pi \setminus$ (app$_\Pi$) $\cup$ (i-app$_\Pi$), $\rightarrow_{\beta\Pi}$

(app$_\Pi$)	$$\frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$
(axiom)	(start) (weak) (Π) (λ)
(conv$_{\beta\Pi}$)	$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_{\beta\Pi} B'}{\Gamma \vdash A : B'}$$
(i-app$_\Pi$)	$$\frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

Lemma:

- If $\Gamma \vdash_\beta A : B$ then $\Gamma \vdash_{\pi_i} A : B$.
- If $\Gamma \vdash_{\pi_i} A : B$ then $\Gamma \vdash_\beta A : [B]_\Pi$
  where $[B]_\Pi$ is the Π-normal form of B.

The π_i-cube

- The π_i-cube loses three of its six properties
  Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_i} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.
  - *We do not have TC* $(\Pi_{y:z}.z)x \not\equiv \Box$ and $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$.
  - *We do not have SR* $(\lambda_{y:z}.y)x \to_{\beta\Pi} x$ but $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$.
  - *We do not have UT2* $\vdash_{\pi_i} * : \Box$, $* =_{\beta\Pi} (\Pi_{z:*}.*)\alpha$, $\alpha : * \vdash_{\pi_i} (\lambda_{z:*}.*)\alpha : (\Pi_{z:*}.*)\alpha$ and $\not\vdash_{\pi_i} (\Pi_{z:*}.*)\alpha : \Box$

- But we have:
  - *We have UT1*
  - *We have STT*
  - *We have PT*
  - *We have SN*
  - *We have a weak form of TC* If $\Gamma \vdash_{\pi_i} A : B$ and *B does not have a Π-redex* then either $B \equiv \Box$ or $\Gamma \vdash_{\pi_i} B : s$.
  - *We have a weak form of SR* If $\Gamma \vdash_{\pi_i} A : B$, *$B$ is not a Π-redex* and $A \twoheadrightarrow_{\beta\Pi} A'$ then $\Gamma \vdash_{\pi_i} A' : B$.

The problem can be solved by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function)

Rules for abbreviations

(start-a)	$\dfrac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A}$	$x \notin \text{DOM}(\Gamma)$
(weak-a)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B}$	$x \notin \text{DOM}(\Gamma)$

Figure 1: Basic abbreviation rules BA

$$(\mathbf{let}_\natural) \qquad \frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\natural_{x:A}.C)B : D[x := B]}$$

Figure 2: (let$_\natural$) where $\natural = \lambda$ or $\natural = \Pi$

The β_a-cube: $R_{\beta_a} = R_\beta$ + BA + let$_\beta$, $\to_\beta$

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$) (conv$_\beta$)

(start-a) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A} \qquad x \notin \text{DOM}(\Gamma)$$

(weak-a) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B} \qquad x \notin \text{DOM}(\Gamma)$$

(let$_\beta$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\lambda_{x:A}.C)B : D[x := B]}$$

Lemma: The β_a-cube satisfies the desirable properties except for typability of subterms.
If A is $\vdash$-legal and B is a subterm of A such that every bachelor $\lambda_{x:D}$ in B is also bachelor in A, then B is $\vdash$-legal.

The π_a-cube: $R_{\pi_a} = R_\pi$ + BA + let$_\beta$ + let$_\Pi$, $\to_{\beta\Pi}$

(axiom) (start) (weak) (Π) (λ) (app$_\Pi$) (conv$_{\beta\Pi}$)

(start-a) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A} \qquad x \notin \mathrm{DOM}(\Gamma)$$

(weak-a) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B} \qquad x \notin \mathrm{DOM}(\Gamma)$$

(let$_\beta$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\lambda_{x:A}.C)B : D[x := B]}$$

(let$_\Pi$) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\Pi_{x:A}.C)B : D[x := B]}$$

Lemma: The π_a-cube satisfies the same properties as the β_a.

The π_{ai}-cube: $R_{\pi_{ai}} = R_{\pi_a} \setminus$ app$_\Pi$ + i-app$_\Pi$, $\to_{\beta\Pi}$

Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_{ai}} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.

- *We NOW have TC* although $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$, we have $\Gamma \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : s$
  By (weak-a) $z : *, x : z, y = x : z \vdash_{\pi_{ai}} z : *$.
  Hence by (let$_\Pi$) $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *[y := x] \equiv *$.

- *We NOW have SR* $(\lambda_{y:z}.y)x \to_{\beta\Pi} x$.
  Although $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$, we have $\Gamma \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$
  Since $z : *, x : z \vdash_{\pi_{ai}} x : z$, and $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *$ and
  $z : *, x : z \Vdash z =_{\beta\Pi} (\Pi_{y:z}.z)x$, we use (conv$_{\beta\Pi}$) to get:
  $z : *, x : z \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$.

Identifying λ and Π

- In the cube of the generalised framework of type systems, we saw that the syntax for terms (functions) and types was intermixed with the only distinction being λ- versus Π-abstraction.

- We unify the two abstractions into one.
$\mathcal{T}_\flat ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{T}_\flat\mathcal{T}_\flat \mid \flat\mathcal{V}{:}\mathcal{T}_\flat.\mathcal{T}_\flat$

- $\mathcal{V}$ is a set of variables and $\boldsymbol{S} = \{*, \Box\}$.

- The β-reduction rule becomes $(\flat)$ $\quad (\flat_{x:A}.B)C \to_\flat B[x := C]$.

- Now we also have the old Π-reduction $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$ which treats type instantiation like function instantiation.

- The type formation rule becomes

$$(\flat_1) \quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\flat x{:}A.B) : s_2} \; (s_1, s_2) \in \boldsymbol{R}$$

(axiom) $$\langle\rangle \vdash * : \Box$$

(start) $$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \; x \notin \text{DOM}(\Gamma)$$

(weak) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B} \; x \notin \text{DOM}(\Gamma)$$

($\flat_2$) $$\frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash (\flat x{:}A.B) : s}{\Gamma \vdash (\flat x{:}A.b) : (\flat x{:}A.B)}$$

(app$\flat$) $$\frac{\Gamma \vdash F : (\flat x{:}A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

(conv) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$$

Isomorphism of the cube and the $\flat$-cube

- For $A \in \mathcal{T}$, we define $\overline{A} \in \mathcal{T}_\flat$ as follows:
  - $\overline{s} \equiv s \qquad \overline{x} \equiv x \qquad \overline{AB} \equiv \overline{A}\,\overline{B}$
  - $\overline{\lambda_{x:A}.B} \equiv \overline{\Pi_{x:A}.B} \equiv \flat_{x:\overline{A}}.\overline{B}$.
- For contexts we define: $\overline{\langle\rangle} \equiv \langle\rangle \quad \overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : \overline{A}$.
- For $A \in \mathcal{T}_\flat$, we define $[A]$ to be $\{A' \in \mathcal{T}$ such that $\overline{A'} \equiv A\}$.
- For context, obviously: $[\Gamma] \equiv \{\Gamma'$ such that $\overline{\Gamma'} \equiv \Gamma\}$.
- If $\Gamma \vdash_\pi A : B$ then $\overline{\Gamma} \vdash_\flat \overline{A} : \overline{B}$.
- If $\Gamma \vdash_\flat A : B$ then there are unique $\Gamma' \in [\Gamma]$, $A' \in [A]$ and $B' \in [B]$ such that $\Gamma' \vdash_\pi A' : B'$.
- *The $\flat$-cube enjoys all the properties of the cube except the unicity of types.*

Recall that extending the cube with Π-reduction loses subject reduction

If we change (appl) by (new appl) in the cube we lose subject reduction.

$$\text{(appl)} \quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

$$\text{(new appl)} \quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

The problem was soolved by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function).

The same problem and solution can be repeated in our $\flat$-cube with type instantiation (Π-reduction).

Adding type instantiation to the typing rules of the ♭-cube

If we change (app♭) by (new app♭) in the ♭-cube we lose subject reduction.

$$\text{(app}\flat\text{)} \quad \frac{\Gamma \vdash_\flat F : (\Pi_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : B[x := a]}$$

$$\text{(app}\flat\flat\text{)} \quad \frac{\Gamma \vdash_\flat F : (\flat_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : (\flat_{x:A}.B)a}$$

Failure of correctness of types and subject reduction

- **Correctness of types no longer holds.** With (appl$\flat\flat$) one can have $\Gamma \vdash A : B$ without $B \equiv \Box$ or $\exists S \,.\, \Gamma \vdash B : S$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ yet $(\flat_{y:z}.z)x \not\equiv \Box$ and $\forall s \,.\, z : *, x : z \not\vdash (\flat_{y:z}.z)x : s$.

- **Subject Reduction no longer holds.** That is, with (appl$\flat$): $\Gamma \vdash A : B$ and $A \twoheadrightarrow A'$ may not imply $\Gamma \vdash A' : B$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\flat_{y:z}.y)x \to_\flat x$, but one can't show $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.

Solving the problem

Keep all the typing rules of the $\flat$-cube the same except: replace (conv) by (new-conv), (appl$\flat$) by (appl$\flat\flat$) and add three new rules as follows:

(start-def) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A} \qquad x \notin \text{DOM}\,(\Gamma)$$

(weak-def) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B} \qquad x \notin \text{DOM}\,(\Gamma)$$

(def) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\flat x{:}A.C)B : D[x := B]}$$

(new-conv) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad \Gamma \vdash B =_{def} B'}{\Gamma \vdash A : B'}$$

(appl$\flat\flat$) $$\frac{\Gamma \vdash F : \flat_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\flat_{x:A}.B)a}$$

In the conversion rule, $\Gamma \vdash B =_{def} B'$ is defined as:

- If $B =_\flat B'$ then $\Gamma \vdash B =_{def} B'$
- If $x = D : C \in \Gamma$ and B' arises from B by substituting one particular free occurrence of x in B by D then $\Gamma \vdash B =_{def} B'$.
- Our 3 new rules and the definition of $\Gamma \vdash B =_{def} B'$ are trying to re-incorporate low-level aspects of functions that are not present in Church's λ-calculus.
- *In fact, our new framework is closer to Frege's abstraction principle and the principles $*$9·14 and $*$9·15 of [Whitehead and Russell, 1910^1, 1927^2].*

Correctness of types holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and want that for some s, $z : *, x : z \vdash (\flat_{y:z}.z)x : s$.
- Here is how the latter formula now holds:

$$
\begin{array}{ll}
z : *, x : z \vdash z : * & \text{(start and weakening)} \\
z : *, x : z.y = x : z \vdash z : * & \text{(weakening)} \\
z : *, x : z \vdash (\flat_{y:z}.z)x : *[y := x] \equiv * & \text{(def rule)}
\end{array}
$$

Subject Reduction holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \to_\beta x$ and we need to show that $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.
- Here is how the latter formula now holds:

$a.$	$z : *, x : z \vdash x : z$	(start and weakening)
$b.$	$z : *, x : z \vdash (\flat_{y:z}.z)x : *$	(from 1 above)
	$z : *, x : z \vdash x : (\flat_{y:z}.z)x$	(conversion, a, b, and $z =_\beta (\flat_{y:z}.z)x$)

Consequences of unifying λ and Π

- A term can have many distinct types. E.g., in λP we have:

  $$\alpha : * \vdash_\beta (\lambda x{:}\alpha.\alpha) : (\Pi x{:}\alpha.*) \qquad \text{and} \qquad \alpha : * \vdash_\beta (\Pi x{:}\alpha.\alpha) : *$$

  which, when we give up the difference between λ and Π, result in:

  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : [x{:}\alpha]*$ and
  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : *$

- More generally, in AUT-QE we have the dervived rule:

$$\frac{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \quad 0 \le m \le n \qquad (3)$$

This derived rule (3) has the following equivalent derived rule in λP (and hence in the higher systmes like $\lambda P\omega$):

$$\frac{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_n{:}A_n.* \qquad 0 \le m \le n}{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_m{:}A_m.\Pi x_{m+1}{:}A_{m+1}.\cdots\Pi x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_m{:}A_m.*}$$

However, AUT-QE goes further and generalises (3) to a rule of *type inclusion*:

$$\frac{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \quad 0 \le m \le n \qquad (Q)$$

The β_Q-cube = β-cube + (Q_β)

$$(\mathrm{Q}_\beta) \qquad \frac{\Gamma \vdash \lambda^{i:1..k}_{x_i:A_i}.A : \Pi^{i:1..n}_{x_i:A_i}.*}{\Gamma \vdash \lambda^{i:1..m}_{x_i:A_i}.\Pi^{i:m+1..k}_{x_i:A_i}A : \Pi^{i:1..m}_{x_i:A_i}.*} \quad 0 \leq m \leq n,\ A \not\equiv \lambda_{x:B}.C$$

- *Lemma:*
  - *The β_Q-cube enjoys all the properties of the cube except the unicity of types.*
  - *Rule Q_β and rule $(s, \square)$ for $s \in \{*, \square\}$ imply rule $(s, *)$.*
    This means that the type systems $\lambda_{\mathrm{Q}}\underline{\omega}$ and $\lambda_{\mathrm{Q}}\omega$ are equal, and that $\lambda_{\mathrm{Q}}P\underline{\omega}$ and $\lambda_{\mathrm{Q}}P\omega$ are equal as well.

- Unicity of types fails for the β_Q-cube. Take: $A : *, x : \Pi_{y:A}.* \vdash x : \Pi_{y:A}.*$ and hence by Q_β, $A : *, x : \Pi_{y:A}.* \vdash x : *$.

Organised multiplicity of Types

For many type systems, unicity of types is not necessary (e.g. Nuprl).

We have an organised multiplicity of types.

1. If $\Gamma \vdash A : B_1$ and $\Gamma \vdash A : B_2$, then $B_1 \stackrel{\diamond}{=} B_2$.

2. If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $A_1 = A_2$, then $B_1 \stackrel{\diamond}{=} B_2$.

3. If $\Gamma \vdash B_1 : s_1$, $B_1 = B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s_1$.

Bibliography

H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*. PhD thesis, Eindhoven University of Technology, 1977. Published as Mathematical Centre Tracts nr. 83 (Amsterdam, Mathematisch Centrum, 1979).

N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [Nederpelt et al., 1994], pages 73–100.

A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [Heijenoort, 1967], pages 1–82.

J.H. Geuvers. *Logics and Type Systems*. PhD thesis, Catholic University of Nijmegen, 1993.

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts, 1967.

D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

Twan Laan and Michael Franssen. Parameters for first order logic. *Logic and Computation*, 2001.

G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.

R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.

F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society,* 2nd series, 25: 338–384, 1926.

G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.

J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.

B. Russell. Letter to Frege. English translation in [Heijenoort, 1967], pages 124–125, 1902.

B. Russell. *The Principles of Mathematics*. Allen & Unwin, London, 1903.

A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910^1, 1927^2. All references are to the first volume, unless otherwise stated.