A Refinement of Barendregt's Cube with Non-First-Class Functions

Fairouz Kamareddine
Heriot-Watt University
Edinburgh, Scotland

http://www.macs.hw.ac.uk/~fairouz/talks/talks2003/indiana.ps

4 April 2003

Summary

- *General definition of function* is key to Frege's *formalisation of logic* (1879).
- *Self-application of functions* was at the heart of *Russell's paradox* (1902).
- To avoid paradoxes, Russell controlled function application via *type theory*.
- Russell (1908) gives the first type theory: the *Ramified Type Theory* (RTT).
- RTT is used in Russell and Whitehead's *Principia Mathematica* (1910–1912).
- *Simple theory of types* (STT): Ramsey (1926), Hilbert and Ackermann (1928).
- Frege's functions $\neq$ Principia's functions $\neq$ *λ-calculus* functions (1932).

- Church's *simply typed λ-calculus* $\lambda\!\to$ = λ-calculus + STT (1940).
- Both RTT and STT are *unsatisfactory*. Hence, birth of *different type systems*, each with *different functional power*. All based on Church's λ-calculus.
- *Eight influential typed λ-calculi* 1940–1988 unified in *Barendregt's cube*.
- Not all functions need to be *fully abstracted* as in the λ-calculus. For some functions, their values are enough.
- *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.
- We extend the cube of the eight influential type systems with non-first-class functions showing that this allows placing the type systems of ML, LF and Automath more accurately in the hierarchy of types.

Prehistory of Types (Paradox Threats)

- *Types* have *always existed* in mathematics, but not explicited until 1879. Euclid avoided *impossible* situations (e.g., two parallel points) via classes/*types*.

- In formal systems, intuition can't use implicit types to avoid impossibilities.

- In 19th century, controversies in analysis led to mathematical *precision*. (*Cauchy*, *Dedekind*, *Cantor*, *Peano*, *Frege*).

- Frege's *general definition of function* was key to his formalisation of logic.

Abstraction Principle 1.

"If in an expression, [. . .] a simple or a compound sign has one or more occurrences and if we regard that sign as replaceable in all or some of these occurrences by something else (but everywhere by the same thing), then we call the part that remains invariant in the expression *a* function, *and* the replaceable part *the* argument *of the function."*

Prehistory of Types (Begriffsschrift/Grundgesetze)

- *An argument* could be a *number* (as in analysis), a *proposition*, or a *function*.
- Distinguishing *1st- and 2nd-level objects avoids paradox in Begriffsschrift*:

  "As functions are fundamentally different from objects, so also *functions whose arguments are and must be functions* are fundamentally different from *functions whose arguments are objects and cannot be anything else*. I call the latter *first-level*, the former *second-level*."

- In Grundgesetze Frege described arithmetic in an extension of *Begriffsschrift*.
- To avoid *paradox*, he *applied a function to its course-of-values*, not *itself*
- Frege treated *courses-of-values* as *ordinary objects*. Hence, *a function that takes objects as arguments could have its own course-of-values as an argument*.

Russell's Paradox, vicious circle principle

- In 1902, Russell wrote Frege saying he *discovered a paradox* in *Begriffsschrift* using $f(x) = \neg x(x)$, (*Begriffsschrift does not suffer from a paradox*).
- Frege replied: *Russell's derivation was incorrect*, $f(f)$ *is not possible in* Begriffsschrift: $f(x)$ needs *objects* as arguments; functions are not objects.
- Using *courses-of-values*, *Russell's argument gives a paradox in* Grundgesetze

Russell *avoided all possible self-references* by the *"vicious circle principle VCP"*:

"*Whatever involves all of a collection* must not be one *of the collection*."

- VCP implemented by a double hierarchy of *types*: *(simple) types* and *orders*.
- The ideas behind simple types was already explained by Frege.

- Due to problems with RTT, the *(Axiom of Reducibility AR)* was introduced
  "For each formula f, there is a formula g with a *predicative* type such that f and g are (logically) equivalent."

- RTT without AR was considered too restrictive and AR itself was questionned.

- Ramsey distinguishes the logical/syntactical and semantical paradoxes.

- RTT without orders eliminates logical paradoxes. Separating language and meta language eliminates semantical paradoxes. No need for orders.

- Simple Theory of Types *(STT)* is RTT without orders.

- STT is not *Church's $\lambda \rightarrow$*. STT *existed (1926) before λ-calculus (1932)*.

The evolution of functions with Frege 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.

- In *Principia Mathematica* [15]: If, for some a, there is a proposition ϕa, then there is a function $\phi\hat{x}$, and vice versa.
- The function ϕ is not a separate entity but always has an argument.

- Frege denoted the course-of-values (*graph*) of a function $\Phi(x)$ by $\grave{\varepsilon}\Phi(\varepsilon)$.

- $\grave{\varepsilon}\Phi(\varepsilon)$ may have given Russell's $\hat{x}\Phi(x)$ for the class of objects with property Φ.

- According to Rosser, the notation $\hat{x}\Phi(x)$ is the basis of the notation $\lambda x.\Phi$.

- Church wrote $\wedge x\Phi(x)$ for $x \mapsto \Phi(x)$ to distinguish it from the class $\hat{x}\Phi(x)$.

λ-calculus does not fully represent functionalisation

1. Abstraction from a subexpression $2 + 3 \mapsto x + 3$

2. Function construction $x + 3 \mapsto \lambda x.x + 3$

3. Application construction $(\lambda x.x + 3)2$

4. Concretisation to a subexpression $(\lambda x.(x + 3))2 \to 2 + 3$

- cannot abstract only half way: $x + 3$ is not a function, $\lambda x.x + 3$ is.

- cannot apply $x + 3$ to an argument: $(x + 3)2$ does not evaluate to 2+3.

Common features of modern types and functions

- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{y:A}.y)[A := \mathbb{N}]$ *has type* $(A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \to A)\mathbb{N} = (A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$ $\qquad$ $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$
- Write $A \to A$ as $\Pi_{y:A}.A$ when y not free in A.

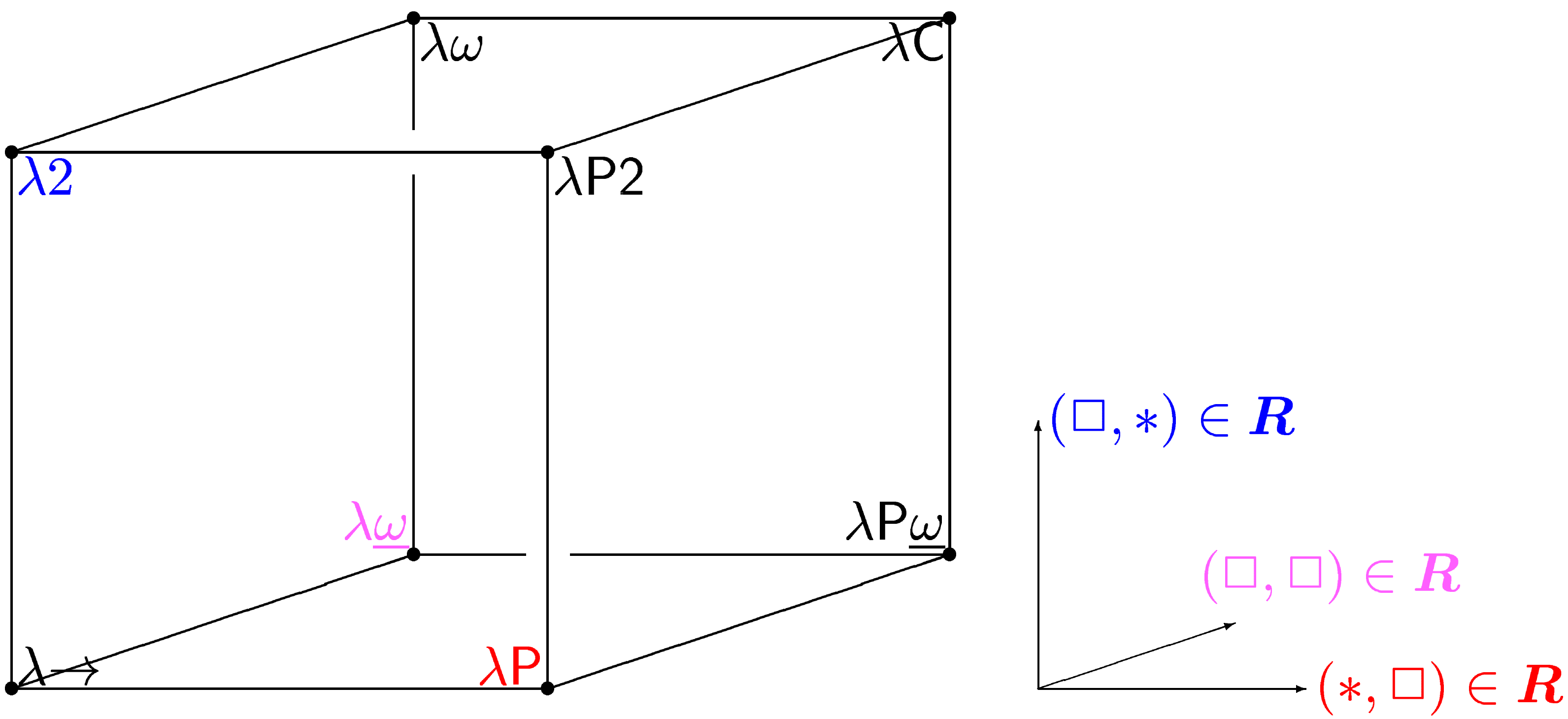
The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$

- Formation rule: $$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2}$$ if $(s_1, s_2) \in \boldsymbol{R}$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\to$	$(*,*)$				λ^τ	[4, 1, 9]
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[7, 14]
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[3, 8]
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[13]
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[11]
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	Fω	[7]
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[5]

The Barendregt Cube



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)

ML

- ML treats let val id $=$ (fn $x \Rightarrow x$) in (id id) end as this Cube term
  $(\lambda \text{id}{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ \text{id}(\beta \to \beta)(\text{id}\,\beta))(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

- 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 $\lambda 2$:
  $(\lambda \text{id} : (\Pi\alpha{:}*.\ \alpha \to \alpha).$
  $\quad(\lambda y{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ y(\beta \to \beta)(y\,\beta))$
  $\quad(\lambda\alpha{:}*.\ \text{id}(\alpha \to \alpha)(\text{id}\,\alpha)))$
  $(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

- Therefore, ML should not have the full Π-formation rule $(\Box, *)$.

- ML has limited access to the rule $(\Box, *)$ enabling some things from $\lambda 2$ but not all.
- ML's type system is none of those of the eight systems of the Cube.
- We place the type system of ML on our refined Cube (between $\lambda 2$ and $\lambda \underline{\omega}$).

LF

- LF [8] is often described as λP of the Barendregt Cube.

- Use of Π-formation rule $(*, \Box)$ is very restricted in the practical use of LF [6].

- The only need for a type $\Pi x{:}A.B : \Box$ is when the Propositions-As-Types principle PAT is applied during the 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 $\Sigma{:}\texttt{prop}$. 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.

- We will find a more precise position of LF on the Cube (between $\lambda\!\to$ and λP).

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 [10].

- 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.

Automath

- The first tool for mechanical representation and verification of mathematical proofs, AUTOMATH, has a parameter mechanism.

- Mathematical text in AUTOMATH written as a finite list of *lines* of the form:

  $x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T.$

  Here g is a new name, an abbreviation for the expression t of type T and $x_1, \ldots, x_n$ are the parameters of g, with respective types $A_1, \ldots, A_n$.

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

- Developments of ordinary mathematical theory in AUTOMATH [2] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

Extending the Cube with parametric constants

- 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 $c(b_1, \ldots, b_n)$.

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

- $\lambda \boldsymbol{R}\boldsymbol{P} = \lambda \boldsymbol{R}$ 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.
- $\lambda \boldsymbol{RP}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.
- Let $\lambda \boldsymbol{RP}$ parametrically conservative (i.e., $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$).
  - The parameter-free system $\lambda \boldsymbol{R}$ is at least as powerful as $\lambda \boldsymbol{RP}$.
  - If $\Gamma \vdash_{\boldsymbol{RP}} a : A$ 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.

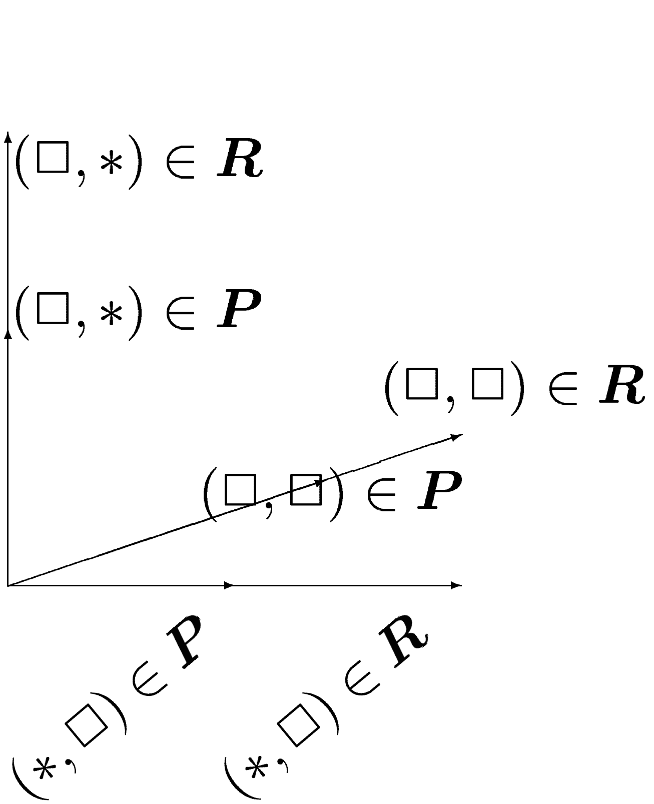
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*), (*,\Box)\}$.

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

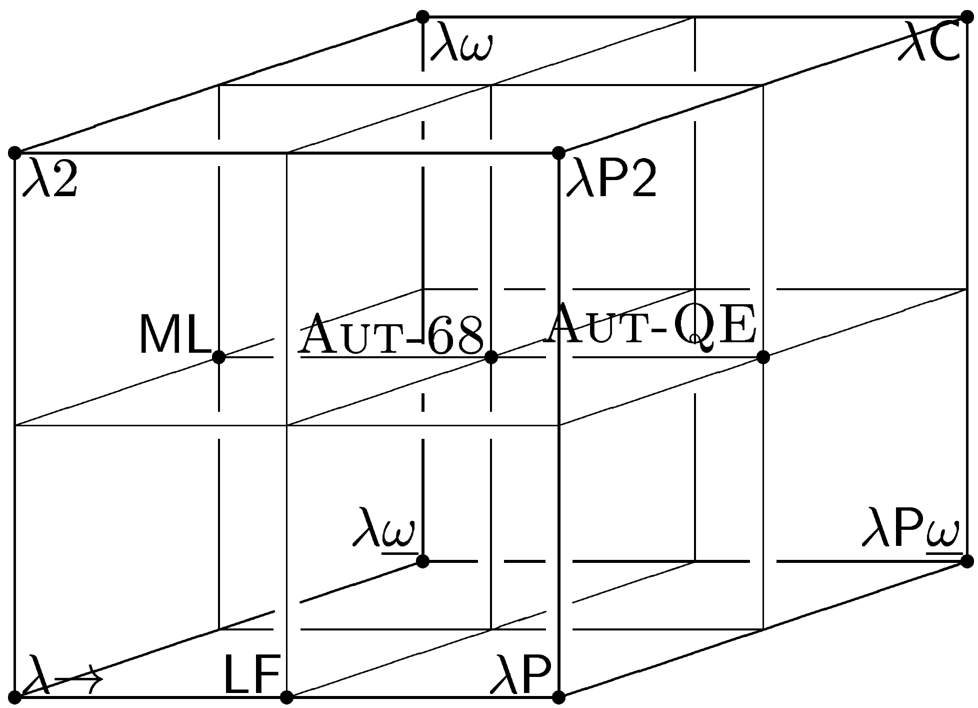
$$\begin{array}{rcl}
\alpha & : & *, \\
\mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\
\mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \;\; . \\
\mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & :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

  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.

Conclusions

- 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 [10].

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

- Parameters enable us to find an exact position of type systems in the generalised framework of type systems.

- Parameters describe the difference between *developers* and *users* of systems.

References

[1] 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.

[2] 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).

[3] 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 [12], pages 73–100.

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

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

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

[15] 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.