Refining the Barendregt Cube with Parameters

Fairouz Kamareddine (Heriot-Watt University, Edinburgh)
Twan Laan and Rob Nederpelt (Eindhoven University of Technology, NL)

7 March 2001

The Low Level approach of functions

- Historically, functions have long been treated as a kind of meta-objects.
- Function *values* have always been important, but abstract functions have not been recognised in their own right until the third of the 20th century.
- In the *low level approach* or *operational* view on functions, there are no functions as such, but only function values.
- E.g., the sine-function, is always expressed together with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- It has long been usual to call $f(x)$—and not f—the function and this is still the case in many introductory mathematics courses.

The revolution of treating functions as first class citizens

- In the nowadays accepted view on functions, they are 'first class citizens'.
- Abstraction and application form the basis of the λ-calculus and type theory.
- This is rigid and does not represent the development of logic in 20th century.
- Frege and Russell's conceptions of functional abstraction, instantiation and application do not fit well with the λ-calculus approach.
- In *Principia Mathematica* [Whitehead and Russell, 1910^1, 1927^2]: 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.

λ-calculus does not fully represent functionalisation

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

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

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

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

- Cannot identify the original term from which a function has been abstracted.

$$\mathtt{let}\ \mathtt{add}_2 = (\lambda x.x + 2)\ \mathtt{in}\ \mathtt{add}_2(x) + \mathtt{add}_2(y)$$

- 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.

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 worthwile 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.

- The representation of a mathematical text in AUTOMATH consists of a finite list of *lines* where every line has the following format:

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

  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 [Benthem Jutting, 1977] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

The Barendregt Cube

- $\mathcal{T}_P ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{T}_P\mathcal{T}_P \mid \lambda\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P \mid \Pi\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P$
- $\mathcal{V}$ is a set of variables and $\boldsymbol{S} = \{*, \Box\}$.

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

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

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

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

(λ) $$\frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.B) : s}{\Gamma \vdash (\lambda x{:}A.b) : (\Pi x{:}A.B)}$$

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

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

Different type formation conditions

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

- $(\Box, *)$ takes care of polymorphism. $\lambda 2$ is weakest on cube satisfying this.

- $(\Box, \Box)$ takes care of type constructors. $\lambda\underline{\omega}$ is weakest on cube satisfying this.

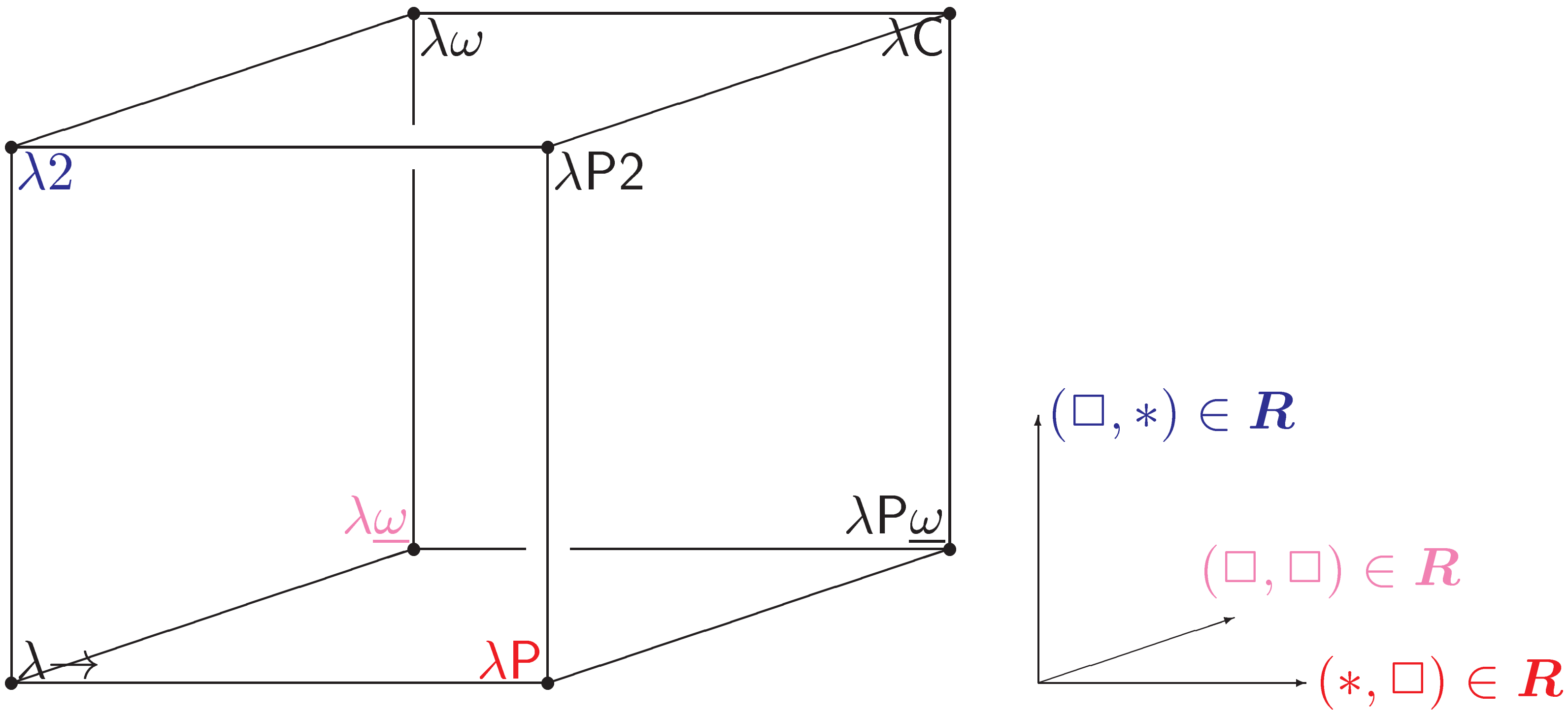
- $(*, \Box)$ takes care of term dependent types. λP is weakest on cube satisfying this.

$\lambda\!\to$	$(*, *)$			
$\lambda 2$	$(*, *)$	$(\Box, *)$		
λP	$(*, *)$		$(*, \Box)$	
$\lambda\underline{\omega}$	$(*, *)$			$(\Box, \Box)$
λP2	$(*, *)$	$(\Box, *)$	$(*, \Box)$	
$\lambda\omega$	$(*, *)$	$(\Box, *)$		$(\Box, \Box)$
λP$\underline{\omega}$	$(*, *)$		$(*, \Box)$	$(\Box, \Box)$
λC	$(*, *)$	$(\Box, *)$	$(*, \Box)$	$(\Box, \Box)$

Systems of the Barendregt Cube

System	Rel. system	Names, references
$\lambda{\to}$	λ^τ	simply typed λ-calculus; [Church, 1940], [Barendregt, 1984] (Appendix A), [Hindley and Seldin, 1986] (Chapter 14)
$\lambda 2$	F	second order typed λ-calculus; [Girard, 1972], [Reynolds, 1974]
λP	AUT-QE	[Bruijn, 1968]
	LF	[Harper et al., 1987]
$\lambda P2$		[Longo and Moggi, 1988]
$\lambda\underline{\omega}$	POLYREC	[Renardel de Lavalette, 1991]
$\lambda\omega$	Fω	[Girard, 1972]
λC	CC	Calculus of Constructions; [Coquand and Huet, 1988]

The Barendregt Cube



LF

- LF (see [Harper et al., 1987]) is often described as λP of the Barendregt Cube.
- [Geuvers, 1993] shows that the use of the Π-formation rule $(*, \Box)$ is very restricted in the practical use of LF.
- This use is in fact based on a parametric construct rather than on Π-formation.
- We will find a more precise position of LF on the Cube (between $\lambda{\to}$ and λP).

ML

- We only consider an explicit version of a subset of ML.
- In ML, One can define the polymorphic identity by:

$$\mathtt{Id}(\alpha{:}*) = (\lambda x{:}\alpha.x) : (\alpha \to \alpha) \tag{1}$$

- But in ML, it is not possible to make an explicit λ-abstraction over $\alpha : *$ by:

$$\mathtt{Id} = (\lambda\alpha{:}*.\lambda x{:}\alpha.x) : (\Pi\alpha{:}*.\alpha \to \alpha) \tag{2}$$

- The type $\Pi\alpha{:}*.\alpha \to \alpha$ does not belong to the language of ML and hence the λ-abstraction of equation (2) is not possible in ML.

ML

- Therefore, we can state that ML does not have a Π-formation rule $(\Box, *)$.
- Nevertheless, ML has some parameter mechanism (α parameter of `Id`)
- ML has limited access to the rule $(\Box, *)$ enabling equation (1) to be defined.
- 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 $\lambda \underline{\omega}$).

Extending the Cube with parametric constructs

- Parametric constructs are $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types.

- $\mathcal{T}_P \ ::= \ \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{C}(\underbrace{\mathcal{T}_{P_1}, \ldots \mathcal{T}_{P_n}}_{n \geq 0}) \mid \mathcal{T}_P \mathcal{T}_P \mid \lambda \mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P \mid \Pi \mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P$

  $\mathcal{C}$ is a set of constants, $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.

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

- $(s_1, s_2) \in \mathbf{P}$ means that we allow parametric constructs $c(b_1, \ldots, b_n) : A$ 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 $(*, s_2) \in \mathbf{P}$ and $(\Box, s_2) \in \mathbf{P}$ 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 $\mathbf{R}$, $\mathbf{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$ containing $(*,*)$.

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

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

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

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

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 $\mathrm{SN}_{\twoheadrightarrow_\beta}(M)$. I.e. M is strongly normalising with respect to $\twoheadrightarrow_\beta$.
- Other properties such as Uniqueness of types and typability of subterms hold.
- $\lambda\mathbf{RP}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.
- Let $\lambda\mathbf{RP}$ parametrically conservative (i.e., $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$).
  - The parameter-free system $\lambda\mathbf{R}$ is at least as powerful as $\lambda\mathbf{RP}$.
  - If $\Gamma \vdash_{\mathbf{RP}} a : A$ then $\{\Gamma\} \vdash_{\mathbf{R}} \{a\} : \{A\}$.

Example

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

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

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

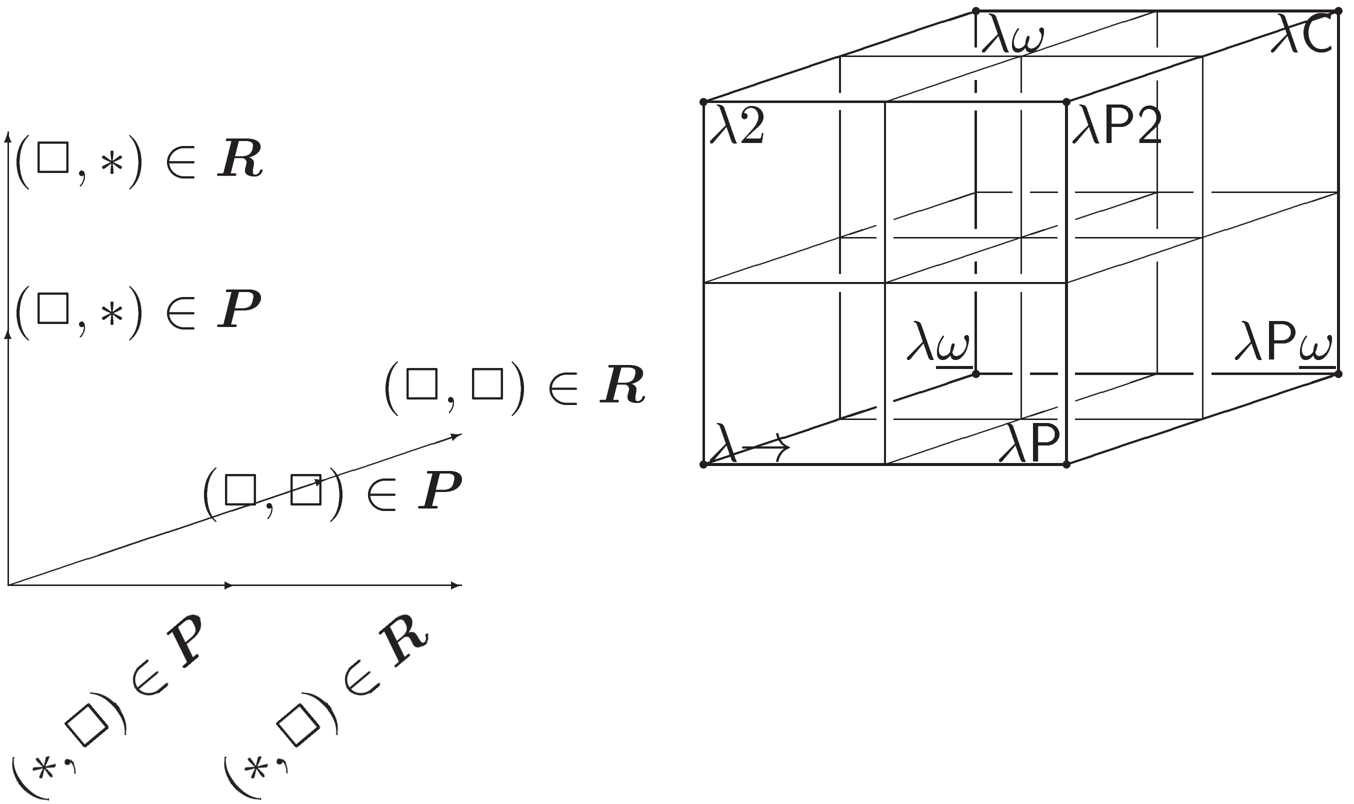
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*),(*,\Box)\}$.

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

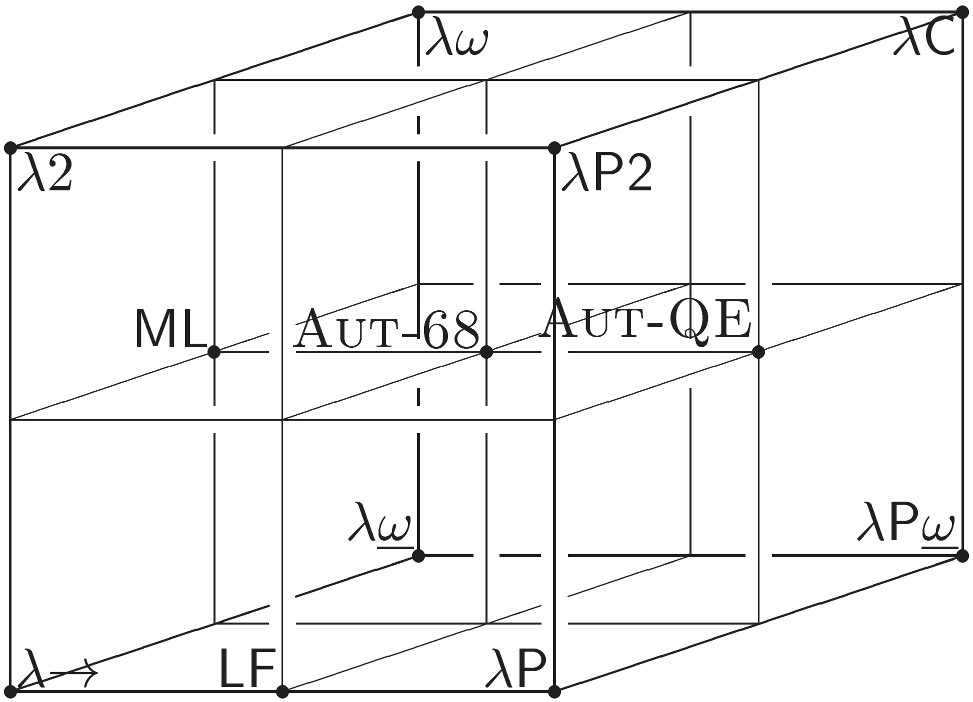
$$
\begin{array}{rcl}
\alpha & : & *, \\
\mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\
\mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \\
\mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & : & \mathtt{eq}(\mathtt{y},\mathtt{x}),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

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

The refined Barendregt Cube



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



LF

- [Geuvers, 1993] initially described LF as the system λP of the Cube.
  However, the Π-formation rule $(*, \Box)$ is restricted in most applications of LF.

- [Geuvers, 1993] splits λ-formation in two (LF – (λ_P) is called LF$^-$):

$$(\lambda_0)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : *}{\Gamma \vdash \lambda_0 x{:}A.M : \Pi x{:}A.B} \qquad (\lambda_0 x{:}A.M)N \to_{\beta_0} M[x{:=}N]$$

$$(\lambda_P)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : \Box}{\Gamma \vdash \lambda_P x{:}A.M : \Pi x{:}A.B} \qquad (\lambda_P x{:}A.M)N \to_{\beta_P} M[x{:=}N]$$

- If $M : *$ or $M : A : *$ in LF, then the β_P-normal form of M contains no λ_P;

- If $\Gamma \vdash_{\mathsf{LF}} M : A$, and Γ, M, A do not contain a λ_P, then $\Gamma \vdash_{\mathsf{LF}^-} M : A$;

- If $\Gamma \vdash_{\mathsf{LF}} M : A(: *)$, all in β_P-normal form, then $\Gamma \vdash_{\mathsf{LF}^-} M : A(: *)$.

LF

- Hence: the only need for a type $\Pi x{:}A.B : \Box$ is to declare a variable in it.

- This is only done when the Propositions-As-Types principle PAT is applied during the construction of the type of the operator Prf as follows:

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

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used.

- No λ_P-abstractions are used. Prf is only used when applied to term $p{:}\texttt{prop}$.

- 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.

- This puts LF in between $\lambda\!\rightarrow$ and λP 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 $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}{\to}*).p0{\to}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm){\to}\Pi n{:}\mathbb{N}.pn \qquad (3)$$

- 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):

$$\texttt{ind}(p{:}\mathbb{N}{\to}*, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn{\to}Snm{\to}pm)) : \Pi n{:}\mathbb{N}.pn \qquad (4)$$

- The logician's type `Ind` is not needed by the mathematician and the types that occur in 4 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 [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.
- 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.

Future Work

- The above only explained the extension of the Cube with parametric constants.
- A larger extension can be made to the more generalised Pure Type Systems.
- We can add definitions and parametric definitions to the Cube and Pure Type systems. This can be found in [Laan, 1997].

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.

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

T. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Eindhoven University of Technology, 1997.

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 Mellon 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.

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.

A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910^1, 1927^2.