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 $\lambda$-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 $\lambda$-calculus approach.

• In *Principia Mathematica* [Whitehead and Russell, 1910\(^1\), 1927\(^2\)]: If, for some $a$, there is a proposition $\phi a$, then there is a function $\phi \bar{x}$, and vice versa.

• The function $\phi$ is not a separate entity but always has an argument.
\(\lambda\text{-calculus does not fully represent functionalisation}\)

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 \((\lambda x . (x + 3))2 \rightarrow 2 + 3\)

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

\[
\text{let } \text{add}_2 = (\lambda x . x + 2) \text{ in } \text{add}_2(x) + \text{add}_2(y)
\]

- 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 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.
and sufficiently readable for humans.

Mechanization is vital for keeping proofs manageable
that this combined definition and parameter
in AUTOMATH [Bezem, Jutting, 1977] revealed
Developments of ordinary mathematical theory

in the context needed for it.
Inherently parameterized by the variables occurring
Each line introduces a new definition which is

parameters of $g$, with respective types $A_1$, $\ldots$, $A_n$, $x$, $\ldots$, $x_i$ are the
expression $f$ of type $L$. $x$, $\ldots$, $x_i$ are the
Here $g$ is a new name, an abbreviation for the

$L : \forall f \in L, x : A_1, \ldots, x_i : A_n \vdash u x : L(f) = (u x, \ldots, u x_i)\ g$

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

has a parameter mechanism.
AUTOMATH verification of mathematical proofs, AUTOMATH.

The first tool for mechanical representation and

Automath

Kamraddeine, Lann and Hendrik
\[
\begin{align*}
\mathcal{B} & : \forall \rightarrow \Gamma \\
\mathcal{B} & \vdash \mathcal{B} & s : \mathcal{B} + \Gamma & \mathcal{B} : \forall \rightarrow \Gamma \\
[\forall = x] \mathcal{B} & : \forall \rightarrow \Gamma & \forall : \forall + \Gamma & (\mathcal{B} \cdot \forall \cdot x) \mathcal{B} + \Gamma \\
(\mathcal{B} \cdot \forall \cdot x) \mathcal{B} + \Gamma & : \forall \rightarrow \Gamma & s : (\mathcal{B} \cdot \forall \cdot x) \mathcal{B} + \Gamma & \mathcal{B} : q + \forall \cdot x \\

\mathcal{R} & \in \{ 2 \} & \mathcal{R} & \in \{ 2 \} & \mathcal{R} : \forall \rightarrow \Gamma & \mathcal{R} : \forall \rightarrow \Gamma \\
(\Gamma) & \text{dom} \not\ni x & \mathcal{B} : \forall + \forall \cdot x , \Gamma & s : \forall + \Gamma & \mathcal{B} : \forall + \Gamma \\
(\Gamma) & \text{dom} \not\ni x & \mathcal{B} : \forall + \forall \cdot x , \Gamma & s : \forall + \Gamma \\
\square & : * \rightarrow \Gamma & \{ \square, * \} & = S & \text{is a set of variables and \forall} \\
\text{The Barendregt Cube} & & & & \text{axiom}
\end{align*}
\]
<table>
<thead>
<tr>
<th>(\square, \square)</th>
<th>(\square, \ast)</th>
<th>(\ast, \square)</th>
<th>(\ast, \ast)</th>
<th>(\overline{\chi})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\square, \square)</td>
<td>(\square, \ast)</td>
<td>(\ast, \square)</td>
<td>(\ast, \ast)</td>
<td>(\overline{\chi})</td>
</tr>
<tr>
<td>(\square, \ast)</td>
<td>(\ast, \square)</td>
<td>(\ast, \ast)</td>
<td>(\overline{\chi})</td>
<td></td>
</tr>
<tr>
<td>(\ast, \ast)</td>
<td>(\ast, \ast)</td>
<td>(\overline{\chi})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \begin{array}{c}
\text{The weakest on cube satisfying this.}
\end{array} \]

\[ \begin{array}{c}
\text{\((\square, \square)\) takes care of term dependent types. \(\chi\) is}
\end{array} \]

\[ \begin{array}{c}
\text{\((\square, \ast)\) takes care of term constructors. \(\overline{\chi}\) is}
\end{array} \]

\[ \begin{array}{c}
\text{\((\ast, \ast)\) takes care of type constructors. \(\chi\) is weakest}
\end{array} \]

\[ \begin{array}{c}
\text{on cube satisfying this.}
\end{array} \]

\[ \begin{array}{c}
\text{\((\square, \square)\) takes care of polymorphism. \(\chi\) is weakest}
\end{array} \]

\[ \begin{array}{c}
\text{Different type formation conditions}
\end{array} \]

\[ \begin{array}{c}
\text{Kamradt, Lamm, and Neederperl}
\end{array} \]
<table>
<thead>
<tr>
<th>Names, References</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculus of Constructions; [Girard, 1972]</td>
<td>F</td>
</tr>
<tr>
<td>Renardel de Lavellette, 1991</td>
<td></td>
</tr>
<tr>
<td>[Longo and Moggi, 1988]</td>
<td></td>
</tr>
<tr>
<td>Harper et al., 1987</td>
<td></td>
</tr>
<tr>
<td>[Briini, 1968]</td>
<td></td>
</tr>
<tr>
<td>[Reynolds, 1974]</td>
<td></td>
</tr>
<tr>
<td>Calculus; [Girard, 1972], [1974]; second order typed $\lambda$-calculus (Chapter 14 and Seligman, 1986 (Chapter 1984 (Appendix A)), [Hindley, Church, 1940], [Barendregt, simp]; typed $\lambda$-calculus;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System of the Barendregt Cube</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kamareddine, Lønn and Nederpelt</td>
<td></td>
</tr>
</tbody>
</table>
The Barendregt Cube

$(\square, \star) \in R$

$(\square, \square) \in R$

$(\star, \square) \in R$
• LF (see [Harper et al., 1987]) is often described as $\lambda P$ of the Barendregt Cube.

• [Geuvers, 1993] shows that the use of the II-formation rule ($\ast, \square$) is very restricted in the practical use of LF.

• This use is in fact based on a parametric construct rather than on II-formation.

• We will find a more precise position of LF on the Cube (between $\lambda \rightarrow$ and $\lambda P$).
We only consider an explicit version of a subset of ML.

In ML, one can define the polymorphic identity by:

\[ \text{Id}(\alpha:* ) = (\lambda x: \alpha . x) : (\alpha \to \alpha) \]  

(1)

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

\[ \text{Id} = (\lambda \alpha : * . \lambda x: \alpha . x) : (\Pi \alpha: * . \alpha \to \alpha) \]  

(2)

The type \( \Pi \alpha: * . \alpha \to \alpha \) does not belong to the language of ML and hence the \( \lambda \)-abstraction of equation (2) is not possible in ML.
Therefore, we can state that ML does not have a \( \Pi \)-formation rule \( (\Box, \ast) \).

Nevertheless, ML has some parameter mechanism \( (\alpha \text{ parameter of } \text{Id}) \)

ML has limited access to the rule \( (\Box, \ast) \) 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 \( \lambda \omega \)).
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.

- \( \mathcal{T}_P ::= \mathcal{V} \mid S \mid \mathcal{C}(\mathcal{T}_{P_1}, \ldots, \mathcal{T}_{P_n}) \mid \mathcal{T}_P \mathcal{T}_P \mid \lambda \mathcal{V} : \mathcal{T}_P. \mathcal{T}_P \mid \Pi \mathcal{V} : \mathcal{T}_P. \mathcal{T}_P \)

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

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

- \( (s_1, s_2) \in \mathcal{P} \) 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 \( (*, s_2) \in \mathcal{P} \) and \( (\Box, s_2) \in \mathcal{P} \) 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 $\mathbb{R}, \mathbb{P} \subseteq \{(*, *), (\star, \square), (\square, \star), (\square, \square)\}$ containing $(\star, \star)$.

- $\lambda \mathbb{R} \mathbb{P} = \lambda \mathbb{R}$ and the two rules ($\mathbb{C}$-weak) and ($\mathbb{C}$-app):

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

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

  $\frac{\Gamma_1, c(\Delta) : A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j := b_j]_{j=1}^{n}}{\Gamma_1, c(\Delta) : A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j := b_j]_{j=1}^{n}}$

$\Delta \equiv x_1 : B_1, \ldots, x_n : B_n.$

$\Delta_i \equiv x_1 : B_1, \ldots, x_{i-1} : B_{i-1}$
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 $A \rightarrow_\beta A'$ then $\Gamma \vdash A' : B$

- **(Strong Normalisation)** For all $\vdash$-legal terms $M$, we have $\text{SN}_{\rightarrow_\beta}(M)$. i.e. $M$ is strongly normalising with respect to $\rightarrow_\beta$.

- Other properties such as **Uniqueness of types** and **typability of subterms** hold.

- $\lambda\text{RP}$ is the system which has $\Pi$-formation rules $R$ and parameter rules $P$.

- Let $\lambda\text{RP}$ parametrically conservative (i.e., $(s_1, s_2) \in P$ implies $(s_1, s_2) \in R$).
  - The parameter-free system $\lambda\text{R}$ is at least as powerful as $\lambda\text{RP}$.
  - If $\Gamma \vdash_{\text{RP}} a : A$ then $\{\Gamma\} \vdash_{R} \{a\} : \{A\}$. 
Example

- $R = \{(*, *)\}$
  $P_1 = \emptyset \quad P_2 = \{(*, *)\} \quad P_3 = \{(*, \Box)\} \quad P_4 = \{(*, *), (*, \Box)\}$
  All $\lambda R P_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

- $R_5 = \{(*, *)\} \quad P_5 = \{(*, *), (*, \Box)\}$.
  $\lambda \rightarrow < \lambda R_5 P_5 < \lambda P$: we can talk about predicates:

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

$\text{eq}$ not possible in $\lambda \rightarrow$. 
The refined Barendregt Cube

\[(\square, \ast) \in R\]

\[(\square, \ast) \in P\]

\[(\square, \square) \in R\]

\[(\ast, \square) \in P\]

\[(\ast, \ast) \in R\]
LF, ML, Aut-68, and Aut-QE in the refined Cube
**LF**

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

- [Geuvers, 1993] splits $\lambda$-formation in two ($\text{LF} - (\lambda_P)$ is called $\text{LF}^-$):

\[
(\lambda_0) \quad \begin{array}{c}
\Gamma, x:A \vdash M : B \\
\Gamma \vdash \Pi x:A.B : \ast
\end{array}
\quad \begin{array}{c}
\Gamma \vdash \lambda_0 x:A.M : \Pi x:A.B
\end{array} \quad \left(\lambda_0 x:A.M\right)N \rightarrow_{\beta_0} M[x:=N]
\]

\[
(\lambda_P) \quad \begin{array}{c}
\Gamma, x:A \vdash M : B \\
\Gamma \vdash \Pi x:A.B : \Box
\end{array} \quad \begin{array}{c}
\Gamma \vdash \lambda_P x:A.M : \Pi x:A.B
\end{array} \quad \left(\lambda_P x:A.M\right)N \rightarrow_{\beta_P} M[x:=N]
\]

- If $M : \ast$ or $M : A : \ast$ in LF, then the $\beta_P$-normal form of $M$ contains no $\lambda_P$;

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

- If $\Gamma \vdash_{\text{LF}} M : A(: \ast)$, all in $\beta_P$-normal form, then $\Gamma \vdash_{\text{LF}^-} M : A(: \ast)$. 
**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 \textsc{pat} is applied during the construction of the type of the operator \textsc{Prf} as follows:

  \[
  \frac{\text{prop} : \star \vdash \text{prop} : \star \quad \text{prop} : \star, \alpha : \text{prop} \vdash \star : \Box}{\text{prop} : \star \vdash (\Pi \alpha : \text{prop}.\star) : \Box}.
  \]

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

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

- Hence, the practical use of LF would not be restricted if we present \textsc{Prf} in a parametric form, and use $(\star, \Box)$ as a parameter instead of a $\Pi$-formation rule.

- This puts LF in between $\lambda \rightarrow$ and $\lambda P$ in the Refined Cube.
Logicians versus mathematicians and induction over numbers

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

\[
\text{Ind} = \Pi p : (\mathbb{N} \to *). p0 \to (\Pi n : \mathbb{N} . \Pi m : \mathbb{N} . \text{Pn} \to Snm \to pm) \to \Pi n : \mathbb{N} . \text{Pn} \quad (3)
\]

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

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

\[
\text{ind}(p : \mathbb{N} \to *, q : p0, r : (\Pi n : \mathbb{N} . \Pi m : \mathbb{N} . \text{Pn} \to Snm \to pm)) : \Pi n : \mathbb{N} . \text{Pn} \quad (4)
\]

- The logician’s type \( \text{Ind} \) is not needed by the mathematician and the types that occur in 4 can all be constructed in \( \lambda R \) with \( R = \{(*, *)(*, \square)\} \).
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).

- \((\Box, \ast)\) is not needed by the mathematician. It is needed in logician’s approach in order to form the \(\Pi\)-abstraction \(\Pi p:(\mathbb{N} \to \ast). \cdots\).

- 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


Kamareddine, Laan and Nederpelt


FLOPS'01, 7–9 March 2001, Waseda, Japan
Kamareddine, Laan and Nederpelt

