Refining the Barendregt Cube using Parameters*

Fairouz Kamareddine**[1], Twan Laan[2], and Rob Nederpelt[3]

[1] Computing and Electrical Engineering, Heriot-Watt Univ., Riccarton, Edinburgh EH14 4AS, Scotland, fairouz@cee.hw.ac.uk
[2] Molenstraat 208, 5701 KL Helmond, the Netherlands, twan.laan@wxs.nl
[3] Mathematics and Computing Science, Eindhoven Univ. of Technology, P.O.Box 513, 5600 MB Eindhoven, the Netherlands, r.p.nederpelt@tue.nl

Abstract. The Barendregt Cube (introduced in [3]) is a framework in which eight important typed λ-calculi are described in a uniform way. Moreover, many type systems (like AUTOMATH [18], LF [11], ML [17], and system F [10]) can be related to one of these eight systems. Furthermore, via the propositions-as-types principle, many logical systems can be described in the Barendregt Cube as well (see for instance [9]).
However, there are important systems (including AUTOMATH, LF and ML) that cannot be adequately placed in the Barendregt Cube or in the larger framework of Pure Type Systems. In this paper we add a parameter mechanism to the systems of the Barendregt Cube. In doing so, we obtain a refinement of the Cube. In this refined Barendregt Cube, systems like AUTOMATH, LF, and ML can be described more naturally and accurately than in the original Cube.

1 Introduction

In [3], Barendregt proposes a framework, now often called the Barendregt Cube, in which eight important and well-known type systems are presented in a uniform way. This makes a detailed comparison of these systems possible. The weakest systems of the Cube is Church's simply typed λ-calculus $\lambda\!\!\rightarrow$ [7], and the strongest system is the Calculus of Constructions λC [8]. Girard's well-known System F [10] figures on the Cube between $\lambda\!\!\rightarrow$ and λC. Moreover, via the Propositions-as-Types principle (see [13]), many logical systems can be described in the systems of the Cube, see [9].

In the Cube, we have in addition to the usual λ-abstraction, a type forming operator Π. Briefly, if A is a type, and B is a type possibly containing the variable x, then $\Pi x{:}A.B$ is the type of functions that, given a term $a : A$, output a value of type $B[x := a]$. Here $a : A$ expresses that a is of type A, and $B[x := a]$ means the result of the substitution of a for x in B. If x does

* This work has been partially supported by EPSRC grant numbers GR/L36963 and GR/L15685, by a Royal Society ESEP grant and by the British council and the Dutch research council NWO.
** Kamareddine is grateful to Eindhoven University of Technology for its hospitality and support during the preparation of this article.

not occur in B, then $\Pi x{:}A.B$ is the type of functions from A to B, written $A \to B$. To the Π-abstraction at the level of types corresponds λ-abstraction at the level of objects. Roughly speaking, if M is a term of type B (M and possibly B containing x), then $\lambda x{:}A.M$ is a term of type $\Pi x{:}A.B$. The Cube has two sorts $*$ (the set of types) and $\Box$ (the set of kinds) with $* : \Box$. If $A : *$ (resp. $A : \Box$) we say A is a type (resp. a kind). All systems of the Cube have the same typing rules. What distinguishes one system from another however is the set $\boldsymbol{R}$ of pairs of sorts (s_1, s_2) allowed in the so-called *type-formation* or *Π-formation* rule, simply referred to as the rule (Π). Each system of the Cube has its own set $\boldsymbol{R}$ (which must contain $(*, *)$). A Π-type can only be formed in a specific system of the Cube if rule (Π) is satisfied for some (s_1, s_2) in the set $\boldsymbol{R}$ of that system. The rule (Π) is as follows:

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

Note that as there are only two sorts, $*$ and $\Box$, and as each set $\boldsymbol{R}$ must contain $(*, *)$, there are only eight possible different systems of the Cube. With the rule (Π), an important aspect of the Cube is that it provides a factorisation of the expressive power of the Calculus of Constructions into three features: *polymorphism, type constructors*, and *dependent types*:

- $(*, *)$ is the basic rule that forms types. All type systems of the Cube have this rule.
- $(\Box, *)$ is the rule that takes care of polymorphism. Girard's System (also known as $\lambda 2$) is the weakest system on the Cube that features this rule.
- $(\Box, \Box)$ takes care of type constructors. The system $\lambda\underline{\omega}$ is the weakest system on the Cube that features this rule.
- $(*, \Box)$ takes care of term dependent types. The system λP is the weakest system on the Cube that features this rule.

Figures 1... 3 illustrate the various systems of the Cube.

Many other well-known type systems, like AUTOMATH [18], LF [11], and ML [17] can be more or less related to one of the systems of the Barendregt Cube. However, the relations between systems from "practice", and systems of the Cube are not always perfect. Here are some examples illustrating this point:

Example 1 (AUTOMATH) All the AUTOMATH systems have a relatively restricted typed λ-calculus. But they are more expressive than their λ-calculus suggests at first sight. This is due to a strong parameter mechanism. Even if one removes the typed λ-calculus from AUTOMATH, a quite expressive system "PAL", fully based on parameters, remains. See [18]. On the other hand, both AUT-68 and AUT-QE have been related to the Cube. But the corresponding Cube-systems are too weak to properly describe these AUTOMATH-systems (see below). We will be able to place both AUT-68 and AUT-QE on our refined Cube.

Example 2 (LF) The system LF (see [11]) is often described as the system λP of the Barendregt Cube. However, Geuvers [9] shows that the use of the Π-formation rule $(*, \Box)$ is very restricted in the practical use of LF. We will see that this use is in fact based on a parametric construct rather than on a Π-formation rule. Here again, we will be able to find a more precise position of LF on the Cube which will be the center of the line whose ends are $\lambda\!\!\rightarrow$ and λP.

Example 3 (ML) In ML (see [17]), types are written implicitly à la Curry. For example, instead of writing $\lambda x{:}A.B$, one writes $\lambda x.B$ and the type checker in ML looks for the type. It is well-known however from [4] that the implicit and explicit type schemes can be related. In any case, for the purposes of our paper, we only consider an explicit version of a subset of ML. Furthermore, we do not treat recursive types nor the Y combinator. In ML, one can define the polymorphic identity by:

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

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

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

Those familiar with ML know that the type $\Pi\alpha{:}*.\alpha \rightarrow \alpha$ does not belong to the language of ML and hence the λ-abstraction of equation (2) is not possible in ML. Therefore, we can state that ML does not have a Π-formation rule $(\Box, *)$. Nevertheless, it clearly has some parameter mechanism (α acting as parameter of `Id`) and hence ML has limited access to the rule $(\Box, *)$ enabling equation (1) to be defined. This means that ML's type system is none of those of the eight systems of the Cube. We will find a place for the type system of ML on our refined Cube. That place will be the intersection of the diagonals of the square (of the Barendregt Cube) whose corners are $\lambda\!\!\rightarrow$, $\lambda 2$, $\lambda\underline{\omega}$, and $\lambda\omega$ (cf. Figure 5).

The above examples show that the Barendregt Cube of [4] cannot accommodate well-known and practical type systems in a precise manner. In this paper, we refine the Barendregt Cube by extending it with a parameter mechanism. Such a mechanism allows the construction of terms of the form $c(b_1, \ldots, b_n)$ where c is a constant and $b_1, \ldots . b_n$ are terms. In traditional typed λ-calculus such a term would be written as $cb_1 \ldots b_n$. This last term is constructed step by step. First, c gets typed, then it is applied to b_1, then the result is applied to b_2, and so on. This means that c, cb_1, cb_1b_2, $\ldots$, $cb_1 \ldots b_n$ are all legal terms of the system. Hence, the attempt to internalise the parameter mechanism into typed λ-calculus as described above, is going too far. In the parametric situation, only $c(b_1, \ldots, b_n)$ is a term. Partial constructions of this term like $c(b_1, \ldots, b_i)$ (for $i < n$) are not a part of the syntax.

Adding parameters *is* an extension, and a useful one, since parametric constructs occur in many practical systems:

Example 4 As explained in Example 1, AUTOMATH has a parametric system.

Example 5 First-order predicate logic has no λ-calculus. It only has parametric constructs. In [15] it is shown that parametric constructs make it possible to give a description of first-order predicate logic in type theory that is much more accurate than the traditional approach in typed λ-calculus.

Example 6 Parameters occur in many parts of computer science. For example, look at the following Pascal fragment P with the function `double`:

```
function double(z : integer) : integer;
begin
   double := z + z
end;
```

P could be represented by the definition

$$\texttt{double} = (\lambda \texttt{z:Int}.(\texttt{z+z})) : (\texttt{Int} \to \texttt{Int}). \tag{3}$$

Of course, this declaration can imitate the behaviour of the function perfectly well. But the construction has the following disadvantages:

- The declaration has as subterm the type $\texttt{Int} \to \texttt{Int}$. This subterm does not occur in P itself. More general, Pascal does not have a mechanism to construct types of the form $A \to B$. Hence, the representation contains terms that do not occur in Pascal;
- `double` itself is not a separate expression in Pascal: you can't write $x :=$ `double` in a program body. One may only use the expression `double` in a program, if one specifies a parameter p that serves as an argument of `double`.

We conclude that the translation of P as given above is not fully to the point. A parameter mechanism allows us to translate P in the parametric form

$$\texttt{double}(z : \texttt{Int}) = (z + z) : \texttt{Int}. \tag{4}$$

This declaration in (4) does not have the disadvantages of (3) described above.

So for an optimal description of practical systems it may be an advantage to study the "mild" extension with parametric constructs only.

In Section 2, we give a short description of the Barendregt Cube. In Section 3, we extend the syntax of the Cube with parametric constructs, and propose types systems that can type these new constructs. In Section 4 we show that the proposed extension in fact leads to a refinement of the Barendregt Cube: it is split into eight smaller cubes. Section 5 places systems like LF, ML, and AUTOMATH in the Refined Barendregt Cube. We conclude in Section 6.

2 The Barendregt Cube

In this section we shortly repeat the definition of the systems in the Cube. For background information the reader may consult [4].

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

Fig. 1. Different type formation conditions

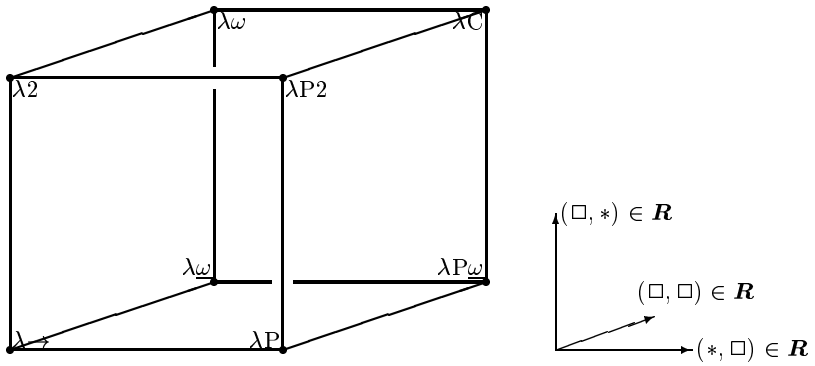

Fig. 2. The Barendregt Cube

System	Related system	Names, references
$\lambda\!\rightarrow$	λ^{τ}	simply typed λ-calculus; [7], [2] (Appendix A), [12] (Chapter 14)
$\lambda 2$	F	second order typed λ-calculus; [10], [20]
λP	AUT-QE	[6]
	LF	[11]
λP2		[16]
$\lambda\underline{\omega}$	POLYREC	[19]
$\lambda\omega$	Fω	[10]
λC	CC	Calculus of Constructions; [8]

Fig. 3. Systems of the Barendregt Cube

Definition 7 (Terms) Let $\mathcal{V}$ be a set of variables. The set $\mathcal{T}$ of terms is defined by the following abstract syntax: $\mathcal{T} ::= * \mid \Box \mid \mathcal{V} \mid \lambda\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \Pi\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \mathcal{T}\mathcal{T}$.

Definition 8 (Contexts) A context is a finite (and possibly empty) list $x_1 : A_1, \ldots, x_n : A_n$ (shorthand: $\vec{x}{:}\vec{A}$) of declarations of typed variables where x_i has type A_i. The set $\{x_1, \ldots, x_n\}$ of distinct variables is called the *domain* $\textsc{dom}\left(\vec{x}{:}\vec{A}\right)$ of the context. The *empty context* is denoted $\langle\rangle$. We use Γ, Δ as meta-variables for contexts.

Definition 9 (Systems of the Barendregt Cube) Let $\mathbf{R}$ be a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$ such that $(*,*) \in \mathbf{R}$. The type system $\lambda\mathbf{R}$ describes in which ways judgments $\Gamma \vdash_{\mathbf{R}} A : B$ (or $\Gamma \vdash A : B$, if it is clear which $\mathbf{R}$ is used) can be derived. $\Gamma \vdash A : B$ states that A has type B in context Γ. The typing rules are inductively defined as follows:

$$\text{(axiom)} \quad \langle\rangle \vdash * : \Box$$

$$\text{(start)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \qquad x \notin \textsc{dom}(\Gamma)$$

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

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

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

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

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

There are eight different possibilities for $\mathbf{R}$ leading to the systems in Figure 1.

The dependencies between these systems can be depicted in the Barendregt Cube (see Figure 2). Furthermore, the systems in the Cube are related to other type systems as is shown in the overview of Figure 3 which is taken from [4].

3 Parameters

We extend the eight systems of the Barendregt Cube with parametric constructs. Parametric constructs are of the form $c(b_1, \ldots, b_n)$ where $b_1, \ldots, b_n$ are terms of certain prescribed types. Just as we can allow several kinds of Π-constructs (via the set $\mathbf{R}$) in the Barendregt Cube, we can also allow several kinds of parametric constructs. This is indicated by a set $\mathbf{P}$, consisting of tuples (s_1, s_2) where $s_1, s_2 \in \{*, \Box\}$. $(s_1, s_2) \in \mathbf{P}$ 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. However, if both $(*, s_2) \in \mathbf{P}$ and $(\Box, s_2) \in \mathbf{P}$ then combinations of parameters are possible. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\Box$.

First we describe the extended syntax.

Definition 10 The set $\mathcal{T}_P$ of *parametric terms* is defined together with the set $\mathcal{L}_V$ of *lists of variables* and the set $\mathcal{L}_T$ of *lists of terms* as follows:

$$\mathcal{T}_P ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{C}(\mathcal{L}_T) \mid \mathcal{T}_P\mathcal{T}_P \mid \lambda\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P \mid \Pi\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P;$$
$$\mathcal{L}_T ::= \varnothing \mid \langle \mathcal{L}_T, \mathcal{T}_P \rangle.$$

where, as usual, $\mathcal{V}$ is a set of variables, $\mathcal{C}$ is a set of constants, and $\boldsymbol{S} = \{*, \Box\}$ is a set of sorts. Formally, lists of terms are of the form $\langle \ldots \langle \langle \varnothing, A_1 \rangle, A_2 \rangle \ldots A_n \rangle$. We usually write $\langle A_1, \ldots, A_n \rangle$ or even $A_1, \ldots, A_n$. In a parametric term of the form $c(b_1, \ldots, b_n)$, the subterms $b_1, \ldots, b_n$ are called the *parameters* of the term.

Let $\vec{x}{:}\vec{A}$ denote $x_1{:}A_1, \ldots, x_n{:}A_n$. We extend the usual definition of $\mathrm{FV}(A)$, the set of *free variables* of a term A, to parametric terms:

$$\mathrm{FV}(c(a_1, \ldots, a_n)) = \textstyle\bigcup_{i=1}^{n} \mathrm{FV}(a_i);$$

Convention 11 Names of bound variables and constants will always be chosen such that they differ from the free ones in a term.

Hence, we do not write $(\lambda \mathtt{x}{:}A.\mathtt{x})\mathtt{x}$ but $(\lambda \mathtt{y}{:}A.\mathtt{y})\mathtt{x}$.

We extend the definition of substitution of a term a for a variable x in a term b, $b[x{:=}a]$, to parametric terms, assuming that x is not a bound variable of either b or a:

$$c(b_1, \ldots, b_n)[x{:=}a] \equiv c(b_1[x{:=}a], \ldots, b_n[x{:=}a]);$$

Definition 12 Given the set of parametric terms, we define the set $\mathcal{C}_P$ of *parametric contexts* (which we denote by $\Gamma, \Gamma', \ldots$) and the set $\mathcal{L}_D$ of *lists of variable declarations* as follows:

$$\mathcal{C}_P ::= \varnothing \mid \langle \mathcal{C}_P, \mathcal{V}{:}\mathcal{T}_P \rangle \mid \langle \mathcal{C}_P, \mathcal{C}(\mathcal{L}_V){:}\mathcal{T}_P \rangle$$
$$\mathcal{L}_D ::= \varnothing \mid \langle \mathcal{L}_D, \mathcal{V}{:}\mathcal{T}_P \rangle.$$

Notice that $\mathcal{L}_D \subseteq \mathcal{C}_P$: all lists of variable declarations are contexts, as well.

Now we extend the typing rules of the Cube as follows:

Definition 13 (The Barendregt Cube with parametric constants) Let $\mathbf{R}$ be as in Definition 9 and let $\mathbf{P}$ be a subset of $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$, such that $(*,*) \in \mathbf{P}$. The judgments that are derivable in $\lambda\mathbf{RP}$ are determined by the rules for $\lambda\mathbf{R}$ of Definition 9 and the following two rules where $\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$ and $\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$:

$$\textbf{($\vec{\mathbf{C}}$-weak)} \quad \frac{\Gamma \vdash b : B \qquad \Gamma, \Delta_i \vdash B_i : s_i \qquad \Gamma, \Delta \vdash A : s}{\Gamma, c(\Delta) : A \vdash b : B} \quad (s_i, s) \in \mathbf{P}$$

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

where the c that is introduced in the $\vec{\text{C}}$-weakening rule is assumed to be Γ-fresh.

At first sight one might miss a $\vec{\text{C}}$-introduction rule. Such a rule, however, is not necessary, as c (on its own) is not a term. c can only be (part of) a term in the form $c(b_1, \ldots, b_n)$, and such terms can be typed by the ($\vec{\text{C}}$-app) rule.

Constant weakening ($\vec{\text{C}}$-weak) explains how we can introduce a declaration of a parametric constant in the context. The context Δ indicates the arity of the parametric constants (the number of declarations in Δ), and of which type each parameter must be ($x_j : B_j$ in Δ means the j-th parameter must be of type B_j).

The extra condition $\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash A : s$ in the ($\vec{\text{C}}$-app) for $n = 0$ is necessary to prevent an empty list of premises. Such an empty list of premises would make it possible to have almost arbitrary contexts in the conclusion. The extra condition is needed to assure that the context in the conclusion is a legal.

A term a is *legal* (with respect to a certain type system) if there are Γ, b such that either $\Gamma \vdash a : b$ or $\Gamma \vdash b : a$ is derivable (in that type system). Similarly, a context Γ is *legal* if there are a, b such that $\Gamma \vdash a : b$.

The parametric type system of Definition 13 has similar meta-theoretical properties as the systems of the Barendregt Cube. We list them below. The proofs are similar to those of the Barendregt Cube (see [14]).

Lemma 14 *Assume $\Gamma \vdash b : B$. Then $\text{DOM}(b), \text{DOM}(B) \subseteq \text{DOM}(\Gamma)$;*

Lemma 15 (Generation Lemma)

1. *If $\Gamma \vdash s : C$ then $s \equiv *$, $C =_\beta \Box$ and if $C \not\equiv \Box$ then $\Gamma \vdash C : s'$ for some sort s'.*
2. *If $\Gamma \vdash x : C$ then there is $s \in \boldsymbol{S}$ and $B =_\beta C$ such that $\Gamma \vdash B : s$ and $(x{:}B) \in \Gamma$;*
3. *If $\Gamma \vdash (\Pi x{:}A.B) : C$ then there is $(s_1, s_2) \in \boldsymbol{R}$ such that $\Gamma \vdash A : s_1$, $\Gamma, x{:}A \vdash B : s_2$ and $C =_\beta s_2$;*
4. *If $\Gamma \vdash (\lambda x{:}A.b) : C$ then there is $s \in \boldsymbol{S}$ and B such that $\Gamma \vdash (\Pi x{:}A.B) : s$; $\Gamma, x{:}A \vdash b : B$; and $C =_\beta (\Pi x{:}A.B)$;*
5. *If $\Gamma \vdash Fa : C$ then there are A, B such that $\Gamma \vdash F : (\Pi x{:}A.B)$, $\Gamma \vdash a : A$ and $C =_\beta B[x{:=}a]$.*
6. *If $\Gamma \vdash c(b_1, \ldots, b_n) : D$ then there exist s, $\Delta \equiv x_1 : B_1, \ldots, x_n : B_n$ and A such that $\Gamma \vdash D =_\beta A[x_j{:=}b_j]_{j=1}^n$, and $\Gamma \vdash b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1}$. Moreover, $\Gamma \equiv \Gamma_1, c(\Delta) : A, \Gamma_2$ and $\Gamma_1, \Delta \vdash A : s$. Finally, there are $s_i \in S$ such that $\Gamma, \Delta_i \vdash B_i : s_i$ and $(s_i, s) \in \mathbf{P}$.*

Lemma 16 (Correctness of Types) *If $\Gamma \vdash A : B$ then $B \equiv s$ or $\Gamma \vdash B : s$ for some $s \in \boldsymbol{S}$.*

Lemma 17 (Subterm Lemma) *If A is legal and B is a subterm of A, then B is legal.*

Lemma 18 (Subject Reduction) *If $\Gamma \vdash A : B$ and $A \to_\beta A'$ then $\Gamma \vdash A' : B$.*

Lemma 19 (Unicity of Types) *If $\Gamma \vdash A : B_1$ and $\Gamma \vdash A : B_2$, then $B_1 =_\beta B_2$.*

Theorem 20 (Strong Normalisation) *If $\Gamma \vdash A : B$ then A and B are β-strongly normalising, that is: any β-reduction path of A or B is finite.*

4 The Refined Barendregt Cube

The systems of Definition 13 have six degrees of freedom: three for the possible choices of $(*, \Box)$, $(\Box, *)$ and $(\Box, \Box) \in \mathbf{R}$ and three for the possible choices of $(*, \Box)$, $(\Box, *)$, and $(\Box, \Box) \in \mathbf{P}$. However, these choices are not independent since constructs that can be made with $\mathbf{P}$-rule (s_1, s_2) can be imitated in a typed λ-calculus with $\mathbf{R}$-rule (s_1, s_2). This means that the parameter-free type system with $\mathbf{R} = \{(*,*), (*, \Box)\}$ is at least as strong as the type system with parameters with the same set $\mathbf{R}$, but with $\mathbf{P} = \{(*,*), (*, \Box)\}$. We make this precise in Theorem 26.

The insight of Theorem 26 can be expressed by depicting the systems with parameters of Definition 13 as a *refinement* of the Barendregt Cube. As in the Barendregt Cube, we start with the system $\lambda\!\!\rightarrow$, which has $\mathbf{R} = \{(*,*)\}$ and $\mathbf{P} = \{(*,*)\}$. Adding an extra element (s_1, s_2) to $\mathbf{R}$ still corresponds to moving in one dimension in the Cube. Now we add the possibility of moving in one dimension in the Cube but stopping half-way. We let this movement correspond to extending $\mathbf{P}$ with (s_1, s_2). This "going only half-way" is in line with the intuition that Π-formation with (s_1, s_2) can imitate the construction of a parametric construct with (s_1, s_2). In other words, the system obtained by "going all the way" is at least as strong as the system obtained by "going only half-way".

The refinement of the Barendregt Cube is depicted in Figure 4. We now make the above intuition that "$\mathbf{R}$ can imitate $\mathbf{P}$" precise.

Definition 21 Consider the system $\lambda\mathbf{RP}$. We call this system *parametrically conservative* if $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$.

Let $\lambda\mathbf{RP}$ be parametrically conservative. In order to show that the parameter-free system $\lambda\mathbf{R}$ is at least as powerful as $\lambda\mathbf{RP}$, we need to remove the parameters from the syntax of $\lambda\mathbf{RP}$. To do so, we replace the parametric application in a term $c(b_1, \ldots, b_n)$ by function application $cb_1, \ldots, b_n$:

Definition 22 Define the parameter-free translation $\{t\}$ of a term $t \in \mathcal{T}_P$ by:

$$\begin{aligned}
\{a\} &\equiv a \quad \text{if } a \equiv x \text{ or } a \equiv s;\\
\{c(b_1, \ldots, b_n)\} &\equiv c\,\{b_1\} \cdots \{b_n\};\\
\{ab\} &\equiv \{a\}\,\{b\};\\
\{\mathcal{O}x{:}A.B\} &\equiv \mathcal{O}x{:}\,\{A\}\,.\,\{B\} \quad \text{if } \mathcal{O} \text{ is } \lambda \text{ or } \Pi
\end{aligned}$$

Definition 23 We extend the definition of $\{\_\}$ to contexts:

$$\begin{aligned}
\{\langle\rangle\} &\equiv \langle\rangle;\\
\{\Gamma, x{:}A\} &\equiv \{\Gamma\}, x{:}\,\{A\};\\
\{\Gamma, c(\Delta){:}A\} &\equiv \{\Gamma\}, c(){:}\,\{\textstyle\prod \Delta.A\}.
\end{aligned}$$

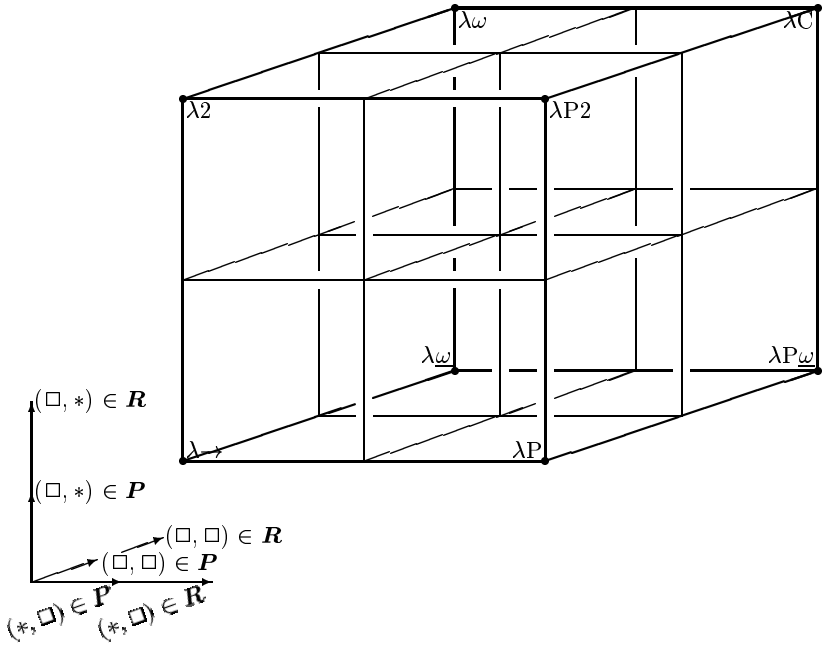

Fig. 4. The refined Barendregt Cube

Here, $\Delta \equiv x_1 : B_1, \ldots, x_n : B_n$, and $\prod \Delta.A$ is shorthand for $\prod_{i=1}^{n} x_n : B_i.A$.

To demonstrate the behaviour of $\{\_\}$ under β-reduction, we need a lemma that shows how to manipulate with substitutions and $\{\_\}$. The proof is straightforward, using induction on the structure of a.

Lemma 24 *For $a, b \in \mathcal{T}_P$: $\{a[x{:=}b]\} \equiv \{a\}\,[x{:=}\{b\}]$.* ⊠

The mapping $\{\_\}$ maintains β-reduction:

Lemma 25 *$a \to_\beta a'$ if and only if $\{a\} \to_\beta \{a'\}$.*

PROOF: Follows easily by induction on the structure of a, and Lemma 24. ⊠

Now we show that $\{\_\}$ embeds the parametrically conservative $\lambda\mathbf{RP}$ in the parameter-free $\lambda\mathbf{R}$:

Theorem 26 *Let $\lambda\mathbf{RP}$ be parametrically conservative. If $\Gamma \vdash_{\mathbf{RP}} a : A$ then $\{\Gamma\} \vdash_{\mathbf{R}} \{a\} : \{A\}$.*

PROOF: Induction on the derivation of $\Gamma \vdash_{\mathbf{RP}} a : A$. By Lemma 24, all cases are easy except for ($\vec{\mathrm{C}}$-weak). So: assume the last step of the derivation was

$$\frac{\Gamma \vdash_{\mathbf{RP}} b : B \qquad \Gamma, \Delta_i \vdash_{\mathbf{RP}} B_i : s_i \qquad \Gamma, \Delta \vdash_{\mathbf{RP}} A : s}{\Gamma, c(\Delta){:}A \vdash_{\mathbf{RP}} b : B} \qquad (s_i, s) \in \boldsymbol{P}.$$

By the induction hypothesis, we have:

$$\{\Gamma\} \vdash_{\mathbf{R}} \{b\} : \{B\}\,; \tag{5}$$
$$\{\Gamma, \Delta_i\} \vdash_{\mathbf{R}} \{B_i\} : s_i; \tag{6}$$
$$\{\Gamma, \Delta\} \vdash_{\mathbf{R}} \{A\} : s. \tag{7}$$

$\lambda\mathbf{RP}$ is parametrically conservative, so $(s_i, s) \in \boldsymbol{R}$ for $i = 1, \ldots, n$. Therefore, we can repeatedly use the Π-formation rule, starting with (7) and (6), obtaining

$$\{\Gamma\} \vdash_{\mathbf{R}} \textstyle\prod_{i=1}^{n} x_i{:}\,\{B_i\}\,.\,\{A\} : s. \tag{8}$$

Notice that $\prod_{i=1}^{n} x_i{:}\,\{B_i\}\,.\,\{A\} \equiv \{\prod \Delta.A\}$. Using ($\vec{\text{C}}$-weak) on (5) and (8) gives

$$\{\Gamma\}, c(){:}\,\{\textstyle\prod \Delta.A\} \vdash_{\mathbf{R}} \{b\} : \{B\}\,. \qquad \boxtimes$$

5 Systems in the Refined Barendregt Cube

In this section, we show that the Refined Barendregt Cube enables us to compare some well-known type systems with systems from the Barendregt Cube. In particular, we show that AUT-68, and AUT-QE, LF, and ML, can be seen as systems in the Refined Barendregt Cube. This is depicted in Figure 5 on page 14, and motivated in the three subsections below.

5.1 AUTOMATH

The AUTOMATH-systems (see [18]) all heavily rely on parametric constructs.
1) AUT-68: The typed λ-calculus of one of the most elementary systems of AUTOMATH, AUT-68, is relatively simple and corresponds to $\lambda{\to}$: it has only $(*, *)$ as a Π-formation rule. This should suggest that AUT-68 has comparable expressiveness $\lambda{\to}$. But for the parametrical constructions there are no limitations in AUT-68 whose parameter mechanism has the following features:

- A line $(\Gamma; k; \text{PN}; \texttt{type})$ in a book is nothing more that the declaration of a parametric constant $k(\Gamma){:}*$. There are no demands on the context Γ, and this means that for a declaration $x{:}A \in \Gamma$ we can have either $A \equiv \texttt{type}$ (in Cube-terminology: $A \equiv *$, so $A : \Box$) or $A{:}\texttt{type}$ (in Cube-terminology: $A : *$). We conclude that AUT-68 has the parameter rules $(*, \Box)$ and $(\Box, \Box)$;
- Similarly, lines of the form $(\Gamma; k; \text{PN}; \Sigma_2)$ where $\Sigma_2{:}\texttt{type}$, represent parametric constants that are constructed using the parameter rules $(*, *)$ and $(\Box, *)$.

This suggests that AUT-68 can be represented by the parametric system with $\mathbf{R} = \{(*, *)\}$ and $\mathbf{P} = \{*, \Box\} \times \{*, \Box\}$. The AUT-68 system can be found in the exact middle of the refined Barendregt Cube.

2) AUT-QE: Something similar holds for the more extensive system AUT-QE. This system has an extra Π-formation rule: $(*, \Box)$ additionally to the rules of

AUT-68. This means that for representing this system, we need the Π-formation rules $\mathbf{R} = \{(*,*),(*,\Box)\}$, and parametric rules (s_1, s_2) for $s_1, s_2 \in \{*, \Box\}$. This system is located in the middle of the right side of the Refined Barendregt Cube, exactly in between λC and λP.

3) PAL: It should be noted that the AUTOMATH languages are all based on two concepts: typed λ-calculus and a parameter/definition mechanism. Both concepts can be isolated: it is possible to study λ-calculus without a parameter/definition mechanism (for instance via the format of Pure Type Systems or the Barendregt Cube of [4]), but one can also isolate the parameter/definition mechanism from AUTOMATH. One then obtains a language that is called PAL, the "Primitive AUTOMATH Language". It cannot be described within the Refined Barendregt Cube (as all the systems in that cube have at least some basic λ-calculus in it), but it can be described as a system with the following parametric specification: $\boldsymbol{R} = \varnothing$; $\boldsymbol{P} = \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.

This parametric specification corresponds to the parametric specifications that were given for the AUTOMATH systems above, from which the Π-formation rules are removed.

5.2 LF

Geuvers [9] initially describes the system LF (see [11]) as the system λP of the Cube. However, the use of the Π-formation rule $(*,\Box)$ is quite restrictive in most applications of LF. Geuvers splits the λ-formation rule in two:

$$(\lambda_0)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : *}{\Gamma \vdash \lambda_0 x{:}A.M : \Pi x{:}A.B};$$

$$(\lambda_P)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : \Box}{\Gamma \vdash \lambda_P x{:}A.M : \Pi x{:}A.B}.$$

System LF without rule (λ_P) is called LF$^-$. β-reduction is split into β_0-reduction and β_P-reduction:

$$(\lambda_0 x{:}A.M)N \to_{\beta_0} M[x{:=}N];$$
$$(\lambda_P x{:}A.M)N \to_{\beta_P} M[x{:=}N].$$

Geuvers then shows that

- If $M : *$ or $M : A : *$ in LF, then the β_P-normal form of M contains no λ_P;
- If $\Gamma \vdash_{\mathrm{LF}} M : A$, and Γ, M, A do not contain a λ_P, then $\Gamma \vdash_{\mathrm{LF}^-} M : A$;
- If $\Gamma \vdash M : A(: *)$, all in β_P-normal form, then $\Gamma \vdash_{\mathrm{LF}^-} M : A(: *)$.

This means that the only real need for a type $\Pi x{:}A.B : \Box$ is to be able to declare a variable in it. The only point at which this is really done is where the bool-style implementation of the Propositions-As-Types principle PAT is made: the construction of the type of the operator Prf (in an unparameterised form) has to be made as follows:

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

In the practical use of LF, this is the only point where the Π-formation rule $(*, \Box)$ is used. No λ_P-abstractions are used, either, and the term Prf is only used when it is applied to a term p:`prop`. This means that the practical use of LF would not be restricted if we introduced Prf in a parametric form, and replaced the Π-formation rule $(*, \Box)$ by a parameter rule $(*, \Box)$. This puts (the practical applications of) LF in between the systems $\lambda\!\!\rightarrow$ and λP in the Refined Barendregt Cube.

5.3 ML

In ML (cf. [17]) one can define the polymorphic identity by (we use the notation of this paper, whereas in ML, the types and the parameters are left implicit):

$$\mathtt{Id}(\alpha{:}*) = (\lambda \mathtt{x}{:}\alpha.\mathtt{x}) : (\alpha \rightarrow \alpha).$$

But we cannot make an explicit λ-abstraction over $\alpha{:}*$. That is, the expression

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

cannot be constructed in ML, as the type $\Pi\alpha{:}*.\alpha \rightarrow \alpha$ does not belong to the language of ML. Therefore, we can state that ML does not have a Π-formation rule $(\Box, *)$, but that it does have the parametric rule $(\Box, *)$.

Similarly, one can introduce the type of lists and some operations by:
$\mathtt{List}(\alpha{:}*) : *$;
$\mathtt{nil}(\alpha{:}*) \ : \mathtt{List}(\alpha)$;
$\mathtt{cons}(\alpha{:}*) : \alpha \rightarrow \mathtt{List}(\alpha) \rightarrow \mathtt{List}(\alpha)$,
but the expression $\Pi\alpha{:}*.*$ does not belong to ML, so introducing `List` by

$$\mathtt{List} : \Pi\alpha{:}*.*$$

is not possible in ML. We conclude that ML does not have a Π-formation rule $(\Box, \Box)$, but only the parametric rule $(\Box, \Box)$. Together with the fact that ML has a Π-formation rule $(*, *)$, this places ML in the middle of the left side of the refined Barendregt Cube, exactly in between $\lambda\!\!\rightarrow$ and $\lambda\omega$.

6 Conclusion

In this paper, we observed that many existing type systems do not fit exactly in the Barendregt Cube. In particular, we explained that previous attempts to describe LF and AUTOMATH were not very successfull. We noted that AUTOMATH uses parameters heavily, and that there are some types that are only used in special situations by LF and that those types and situations could be covered by parameters. In addition, we considered an explicitly typed version of ML and noted that there too, ML cannot occupy any of the corners of the cube. The reason being that, ML (as well as LF and AUTOMATH) allows Π-types, but not all of them. In any corner of the Cube, as soon as an abstraction of a sort is allowed, all abstractions of that sort are allowed too.

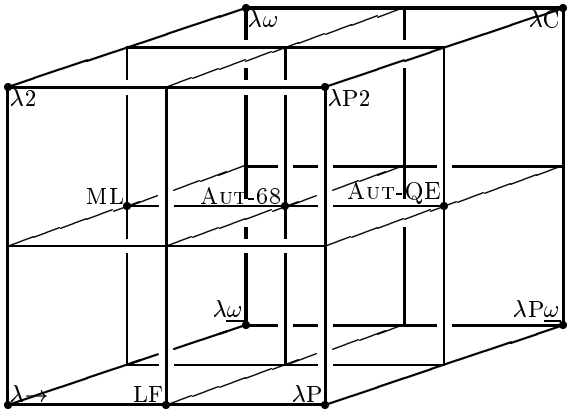

Fig. 5. LF, ML, AUT-68, and AUT-QE in the refined Barendregt Cube

Our above reasoning led us to propose a refinement of the Cube where not only the eight corners can be inhabited, but also points half way between these corners. This way, AUTOMATH, LF, and ML find more accurate locations on the Cube to represent their typing systems. We described an extension of the Barendregt Cube with parameters. This is more a *refinement* than an *extension*, as new systems that are introduced can be depicted by dividing the traditional Barendregt Cube into eight sub-cubes. This is due to the fact that parametric constructs can be imitated by constructions of typed λ-calculus (see Theorem 26) but not the other way around.

We showed that our refinement makes it possible to:

- Give a better description of practical type systems like LF and ML than the systems in the usual Cube.
- Position systems that could not be placed in the usual Cube (several AUTOMATH-systems).

This makes it possible to give a more detailed comparison between the expressiveness of several type systems.

Not only can we add parameters to the Barendregt Cube resulting in an elegant and more refined hierarchy of systems, but we can follow a similar construction to the more generalised notion of Pure Type Systems (PTSs) (see [4]). In addition, we can add definitions (see [5, 22]) and parametric definitions to our above refinement of the Cube and even to the refinements of PTSs, giving a very general hierarchy that can express more precisely and elegantly many practical systems and that give a full description of AUTOMATH.

References

1. S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures.* Oxford

University Press, 1992.
2. 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.
3. H.P. Barendregt. Introduction to generalised type systems. *Functional Programming*, 1:125–154, 1991.
4. H.P. Barendregt. Lambda calculi with types. In [1], pages 117–309. Oxford University Press, 1992.
5. R. Bloo, F. Kamareddine, and R. P. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126 (2):123–143, 1996.
6. 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 [18], pages 73–100.
7. A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
8. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
9. J.H. Geuvers. *Logics and Type Systems*. PhD thesis, Catholic University of Nijmegen, 1993.
10. J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.
11. 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.
12. 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.
13. W.A. Howard. The formulas-as-types notion of construction. In [21], pages 479–490, 1980.
14. T. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Eindhoven University of Technology, 1997.
15. Twan Laan and Michael Franssen. Parameters for first order logic. *Logic and Computation*, 2001.
16. G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.
17. R. Milner, M. Tofte, and R. Harper. *Definition of Standard ML*. MIT Press, Cambridge (Massachusetts)/London, 1990.
18. 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.
19. G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.
20. J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.
21. J.P. Seldin and J.R. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, New York, 1980.
22. P. Severi and E. Poll. Pure type systems with definitions. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of LFCS'94 (LNCS* **813***)*, pages 316–328, New York, 1994. LFCS'94, St. Petersburg, Russia, Springer Verlag.