Two refinements of the λ calculus and the Barendregt Cube: Item Notation and Parameters

Fairouz Kamareddine (Heriot-Watt University)
Based on joint work with Twan Laan and Rob Nederpelt (Eindhoven,NL)

8 January 2001

Item Notation/Lambda Calculus à la de Bruijn

- For those used to classical notation, $\mathcal{I}$ translates to item notation:

$$\mathcal{I}(x) = x, \qquad \mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B), \qquad \mathcal{I}(AB) = (\mathcal{I}(B))\mathcal{I}(A)$$

- For example, $\mathcal{I}((\lambda x.(\lambda y.xy))z) = (z)[x][y](y)x$. The *items* are (z), $[x]$, $[y]$ and (y).

- The *applicator wagon* (z) and *abstractor wagon* $[x]$ occur NEXT to each other.

- In classical notation the β rule is $(\lambda x.A)B \to_\beta A[x := B]$. In item notation, the rule is:

$$(B)[x]A \to_\beta [x := B]A$$

Redexes in Item Notation

Classical Notation	Item Notation
$(\underline{(\lambda_x.(\lambda_y.\lambda_z.zd)c)b})a$	$(a)\underline{(b)[x]}(c)[y][z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$(\underline{(\lambda_y.\lambda_z.zd)c})a$	$(a)\underline{(c)[y]}[z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\lambda_z.zd)a}$	$\underline{(a)[z]}(d)z$
$\downarrow_\beta$	$\downarrow_\beta$
ad	$(d)a$

$$(a)\ (b)\ [x]\ (c)\ [y]\ [z]\ (d)\ \ z$$

Segments, Partners, Bachelors

- The "bracketing structure" of the classical notation $((\lambda_x.(\lambda_y.\lambda_z.--)c)b)a)$, is '$\{_1\ \{_2\ \{_3\ \}_2\ \}_1\ \}_3$', where '$\{_i$' and '$\}_i$' match.

- In item notation, $(a)(b)[x](c)[y][z](d)$ has the simpler bracketing structure $\{\{\ \}\{\ \}\}$.

- An applicator (a) and an abstractor $[x]$ are *partners* when they match like '$\{$' and '$\}$'. Non-partnered items are *bachelors*. A *segment* $\overline{s}$ is *well balanced* when it contains only partnered items.

- Example: Let $\overline{s} \equiv (a)(b)[x](c)[y][z](d)$. Then: The items (a), (b), $[x]$, (c), $[y]$, and $[z]$ are partnered. The item (d) is a bachelor. The segment $(a)(b)[x](c)[y][z]$ is well balanced.

More on Segments, Partners, and Bachelors

Consider some term $\overline{s}x$. Some facts:

- The *main* items in $\overline{s}$ are those at top level, not within some applicator (a).
- Each main bachelor abstractor $[x]$ precedes each main bachelor applicator (a).
- Removing all main bachelor items from $\overline{s}$ yields a well balanced segment.
- Removing all main partnered items from $\overline{s}$ yields a segment $[v_1]\dots[v_n](a_1)\dots(a_m)$ consisting of all main bachelor abstractors followed by all main bachelor applicators.
- If $\overline{s}$ is of the form $\overline{s_1}(b)\overline{s_2}[v]\overline{s_3}$ where $[v]$ and (b) are partnered, then $\overline{s_2}$ must be well balanced.

Even More on Segments, Partners, and Bachelors

Each non-empty segment $\overline{s}$ has a unique *partitioning* into sub-segments $\overline{s} = \overline{s_0}\,\overline{s_1} \cdots \overline{s_n}$ such that

- For even i, the segment $\overline{s_i}$ is well balanced. For odd i, the segment $\overline{s_i}$ is a bachelor segment, i.e., it contains only bachelor main items.
- All well balanced segments after the first and all bachelor segments are non-empty.
- If $\overline{s_i} = [x_1] \cdots [x_m]$ (only abstractor main items) and $\overline{s_j} = (a_1) \cdots (a_p)$ (only applicator main items), then $i < j$, i.e., $\overline{s_i}$ precedes $\overline{s_j}$ in $\overline{s}$.

Example

$\overline{s} \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$, has the following partitioning:

- well-balanced segment $\overline{s_0} \equiv \emptyset$
- bachelor segment $\overline{s_1} \equiv [x][y]$,
- well-balanced segment $\overline{s_2} \equiv (a)[z]$,
- bachelor segment $\overline{s_3} \equiv [x'](b)$,
- well-balanced segment $\overline{s_4} \equiv (c)(d)[y'][z']$,
- bachelor segment $\overline{s_5} \equiv (e)$.

More on Item Notation

- Above discussion and further details of item notation can be found in [Kamareddine and Nederpelt, 1995, 1996].

- Item notation helped greatly in the study of a one-sorted style of explicit substitutions, the λs-style which is related to $\lambda\sigma$, but has certain simplifications [Kamareddine and Ríos, 1997, 1995, 2000].

Canonical Forms

- Item notation helps in finding nice canonical forms. The term

$$[x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$$

is equivalent to

$$[x][y][x'](a)[z](c)(d)[y'][z'](b)(e)$$

and also

$$[x][y][x'](a)[z](d)[y'](c)[z'](b)(e)$$

- Nice canonical forms look like:

bachelor [ ]s	( )[ ]-pairs, A_i in CF	bachelor ( )s, B_i in CF	end var
$[x_1]\ldots[x_n]$	$(A_1)[y_1]\ldots(A_m)[y_m]$	$(B_1)\ldots(B_p)$	x

- In classical notation:

$$\lambda x_1 \cdots \lambda x_n.(\lambda y_1.(\lambda y_2. \cdots (\lambda y_m.x B_p \cdots B_1) A_m \cdots) A_2) A_1$$

Some Rules for Generalising Reduction

Name	In Classical Notation	In Item Notation
(θ)	$((\lambda_x.N)P)Q \downarrow (\lambda_x.NQ)P$	$(Q)(P)[x]N \downarrow (P)[x](Q)N$
(γ)	$(\lambda_x.\lambda_y.N)P \downarrow \lambda_y.(\lambda_x.N)P$	$(P)[x][y]N \downarrow [y](P)[x]N$
(g)	$((\lambda_x.\lambda_y.N)P)Q \downarrow (\lambda_x.N[y:=Q])P$	$(Q)(P)[x][y]N \downarrow (P)[x][y:=Q]N$
(γ_C)	$((\lambda_x.\lambda_y.N)P)Q \downarrow (\lambda_y.(\lambda_x.N)P)Q$	$(Q)(P)[x][y]N \downarrow (Q)[y](P)[x]N$

Obtaining Canonical Forms

The results of going to normal form for the indicated reduction rules, in the order shown:

θ:		$()[\,]$-pairs mixed with bach. $[\,]$s $(A_1)[x][y][z](A_2)[p]\cdots$	bach. $()$s $(B_1)(B_2)\cdots$	end var x
γ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs mixed with bach. $()$s $(B_1)(A_1)[x](B_2)\cdots$		end var x
θ, γ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. $()$s $(B_1)(B_2)\ldots$	end var x
γ, θ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. $()$s $(B_1)(B_2)\ldots$	end var x

More on Canonical Forms

- Both $\theta(\gamma(A))$ and $\gamma(\theta(A))$ are in *canonical form* and we have that $\theta(\gamma(A)) =_p \gamma(\theta(A))$ where $\rightarrow_p$ is the rule

$$(A_1)[y_1](A_2)[y_2]B \rightarrow_p (A_2)[y_2](A_1)[y_1]B \qquad \text{if } y_1 \notin \mathrm{FV}(A_2)$$

- For a term A, we define: $[A] = \{B \mid \theta(\gamma(A)) =_p \theta(\gamma(B))\}$.

- When $B \in [A]$, we write that $B \approx_{\texttt{equi}} A$.

- One-step class-reduction $\leadsto_\beta$ is the least compatible relation such that:

$$A \leadsto_\beta B \quad \text{iff} \quad \exists A' \in [A].\exists B' \in [B].\ A' \to_\beta B'$$

- Classes ($[A]$) and class reduction ($\leadsto_\beta$) nicely preserve various strong normalization properties.

- Define $A \leadsto_\beta^{(E)[x]} B$ iff $\exists A' \in [A].\exists B' \in [B].\exists E' \in [E].\ A' \to_\beta^{(E')[x]} B'$.

Theorem 1. *If* $A \approx_{\texttt{equi}} C$ *and* $A \leadsto_{\beta}^{(E)[x]} B$ *then*
$(\exists D, E')[B \approx_{\texttt{equi}} D, E' \approx_{\texttt{equi}} E,$ *and* $C \rightarrow_{\beta}^{(E')[x]} D]$.

$$
\begin{tikzcd}
A \arrow[rr, rightsquigarrow, "(E)\lbrack x\rbrack"] \arrow[d, phantom, "\approx_{\texttt{equi}}"] & & B \arrow[d, phantom, "\approx_{\texttt{equi}}"] \\
C \arrow[rr, dashed, rightsquigarrow, "(E')\lbrack x\rbrack"] & & D
\end{tikzcd}
$$

A Few Uses of Generalised Reduction and Term Reshuffling

- Regnier [1992] uses term reshuffling and generalized reduction in analyzing perpetual reduction strategies.
- Term reshuffling is used in [Kfoury et al., 1994], [Kfoury and Wells, 1994] in analyzing typability problems.
- [Nederpelt, 1973; de Groote, 1993; Kfoury and Wells, 1995] use generalised reduction and/or term reshuffling in relating SN to WN.
- [Ariola et al., 1995] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- [Kamareddine and Nederpelt, 1995; Bloo et al., 1996] showed how generalized reduction and term reshuffling could reduce space/time needs.
- [Kamareddine, 2000] shows various strong properties of generalised reduction.

What are Parameters?

- Historically, functions have long been treated as a kind of of meta-objects.
- In the nowadays accepted view on functions, they are 'first class citizens'.
- Function *values* have always been important, but abstract functions have not been recognised in their own right until the middle 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: $\sin(2x) = 2\sin(x)\cos(x)$.

What are Parameters?

- 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.

- 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.

- An important difference between the low-level and high-level approach is whether functions are 'spectators' in the world under consideration which can be called upon for services but do not join the ongoing play, or 'participants' standing on stage just like the other players.

Advantages of Parameters

- The corresponding theory can be of lower order than in the high-level case, e.g. first-order with parameters versus second-order without.

- Possible to *fine-tune* a theory by using parameters for some classes of functions.

- 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.

A different form of abstraction and application

- Abstraction and application form the basis of a type system. This view is rigid and does not represent the development of logic in the 20th century.
- Frege and Russell's conceptions of functional abstraction, instantiation and application do not fit well with the λ-calculus approach.
- Here is an example taken from *Principia Mathematica* (cf. [Whitehead and Russell, 1910^1, 1927^2]):

  $*\mathbf{9{\cdot}15}$. 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.

Developers versus users of a type theory

- The parameter mechanism enables us to describe the difference between *developers* and *users* of certain systems.
- Logicians versus mathematicians and the induction axiom for natural numbers.
- A logician is someone developing this axiom (or studying its properties).
- The mathematician is usually only interested in applying (using) the axiom.

The logician and Induction

- Logician: The induction axiom can be described in a PTS with sorts $*, \Box$, axiom $* : \Box$ and Π-formation rules $(*, *, *)$, $(*, \Box, \Box)$, $(\Box, *, *)$ by the PTS-type $\mathtt{Ind}$:

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

  $\mathtt{ind}$: $\mathtt{Ind}$ serves as a proof term for any application of the induction axiom.

The mathematician and Induction

- Mathematician: only *applies* the induction axiom and doesn't need to know the proof-theoretical backgrounds.

- Mathematician uses the term `ind` only in combination with terms 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 $(\mathtt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.

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

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

The mathematician's use of Induction

- The types that occur in this scheme can all be constructed using sorts $*, \Box$, axiom $* : \Box$ and rules $(*, *, *)$, $(*, \Box, \Box)$.
- The rule $(\Box, *, *)$ is not needed (in the logician's approach, this rule was needed 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.

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 $A_1, \ldots, A_n$.

Automath

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

- Actual development of ordinary mathematical theory in the AUTOMATH system by e.g. van Benthem Jutting (cf. [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

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

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

Figure 1: Different type formation conditions

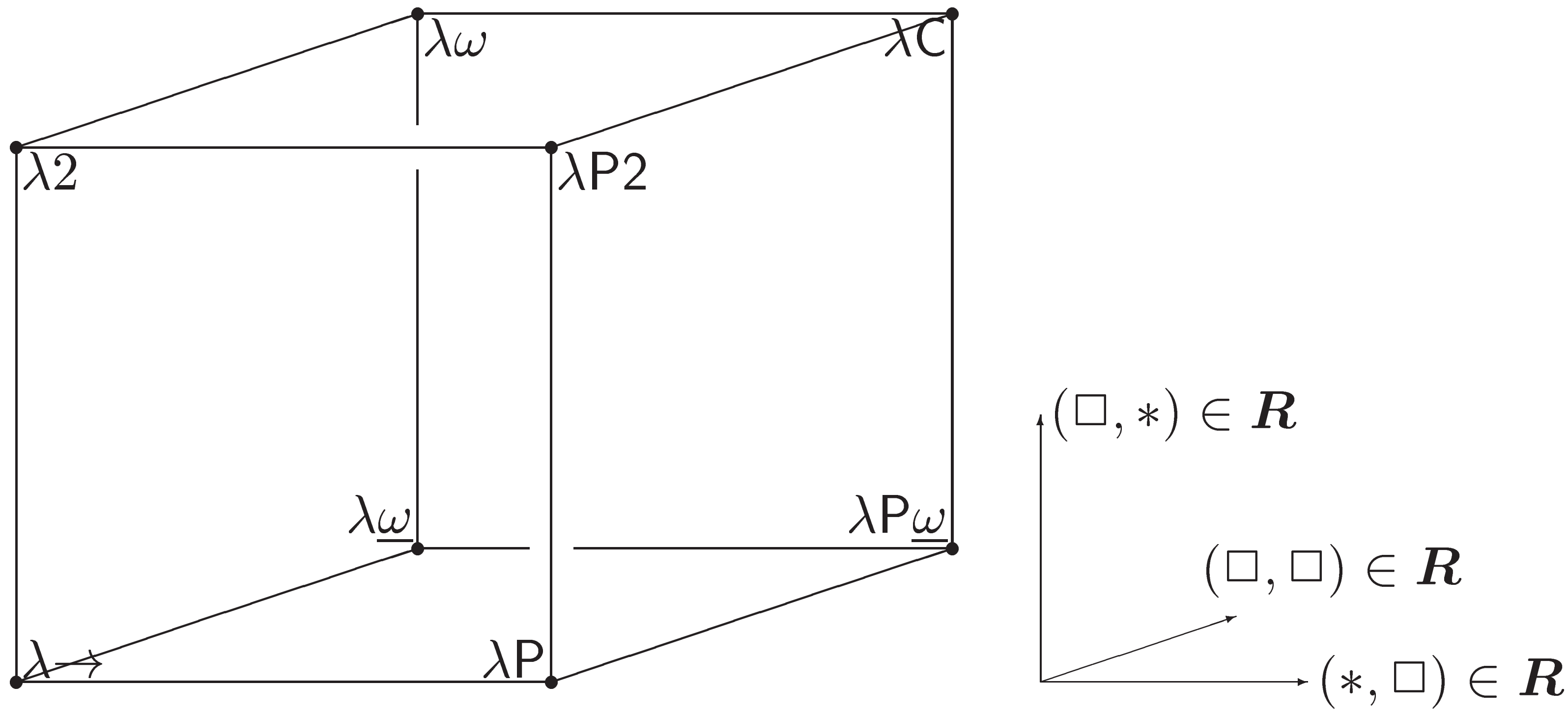

Figure 2: The Barendregt Cube

System	Related system	Names, references
$\lambda{\rightarrow}$	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]
λP2		[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]

Figure 3: Systems of the Barendregt Cube

LF

- The system LF (see [Harper et al., 1987]) is often described as the system λP of the Barendregt Cube.
- However, Geuvers [Geuvers, 1993] 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.

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:

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

- Those familiar with ML know that 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, it clearly has some parameter mechanism (α acting as parameter of `Id`)
- 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.

Extending the Cube with parametric constructs

- 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 $s_1, s_2 \in \{*, \Box\}$.

Extending the Cube with 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.
- 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 $\Box$.

Extending the Cube with parametric constructs

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

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

- $\langle \ldots \langle\langle \varnothing, A_1\rangle, A_2\rangle \ldots A_n\rangle$ is written $\langle A_1, \ldots, A_n\rangle$ 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.

The Barendregt Cube with parametric constants

- Let $\mathbf{R}$, $\mathbf{P}$ be subsets of $\{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$ containing $(*,*)$.
- The judgments that are derivable in $\lambda\mathbf{RP}$ are determined by the usual rules for $\lambda\mathbf{R}$ and the following two rules where $\Delta \equiv x_1{:}B_1,\ldots,x_n{:}B_n$ and $\Delta_i \equiv x_1{:}B_1,\ldots,x_{i-1}{:}B_{i-1}$:

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

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

Lemma 1. *Correctness of types) If $\Gamma \vdash A : B$ then ($B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort S).*

Theorem 1. *(Subject Reduction SR) If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$* $\Box$

Theorem 2. *(Strong Normalisation) For all $\vdash$-legal terms M, we have $SN_{\twoheadrightarrow_\beta}(M)$. I.e. M is strongly normalising with respect to $\twoheadrightarrow_\beta$.*

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

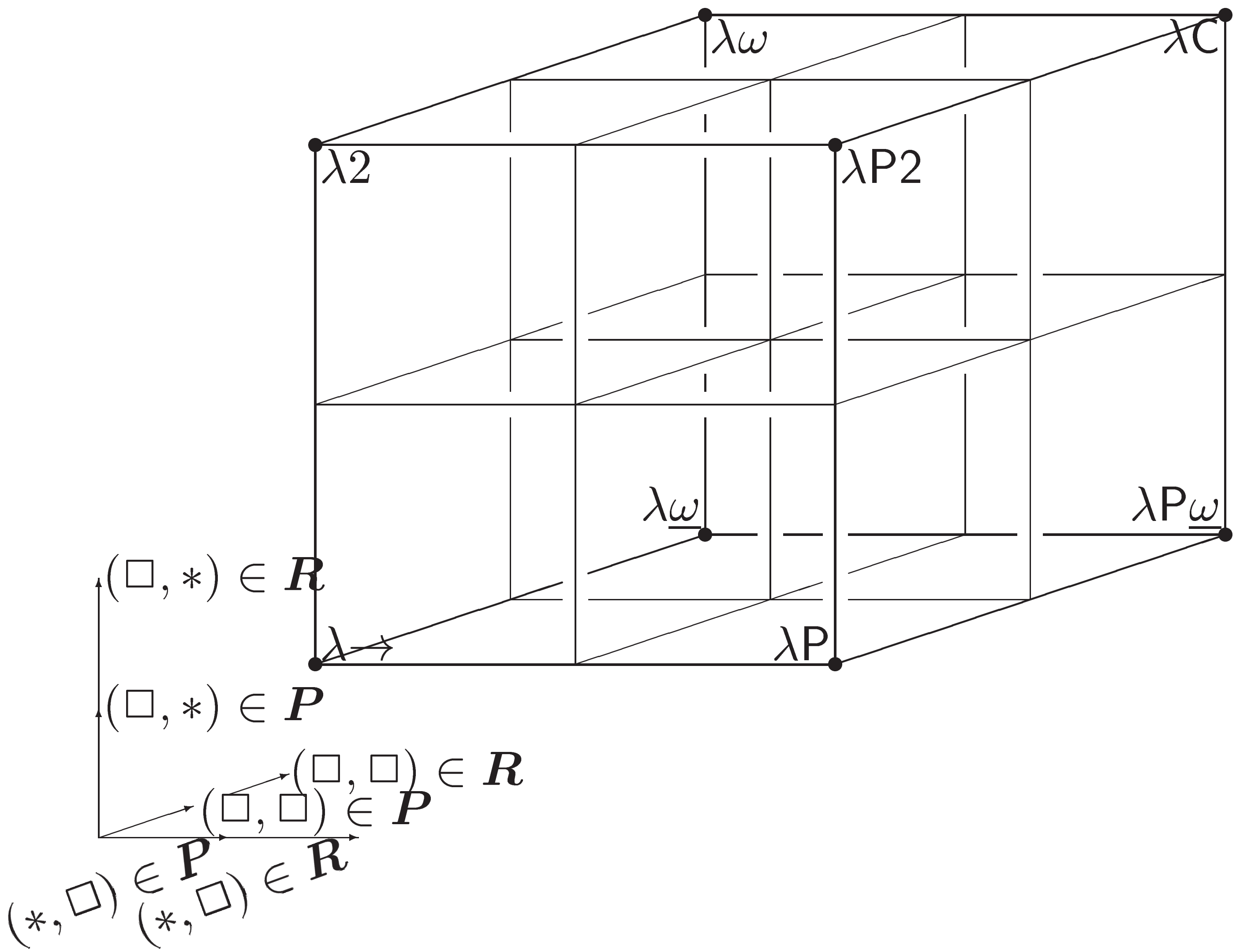

Figure 4: The refined Barendregt Cube

- 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. The parameter-free system $\lambda\mathbf{R}$ is at least as powerful as $\lambda\mathbf{RP}$.

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

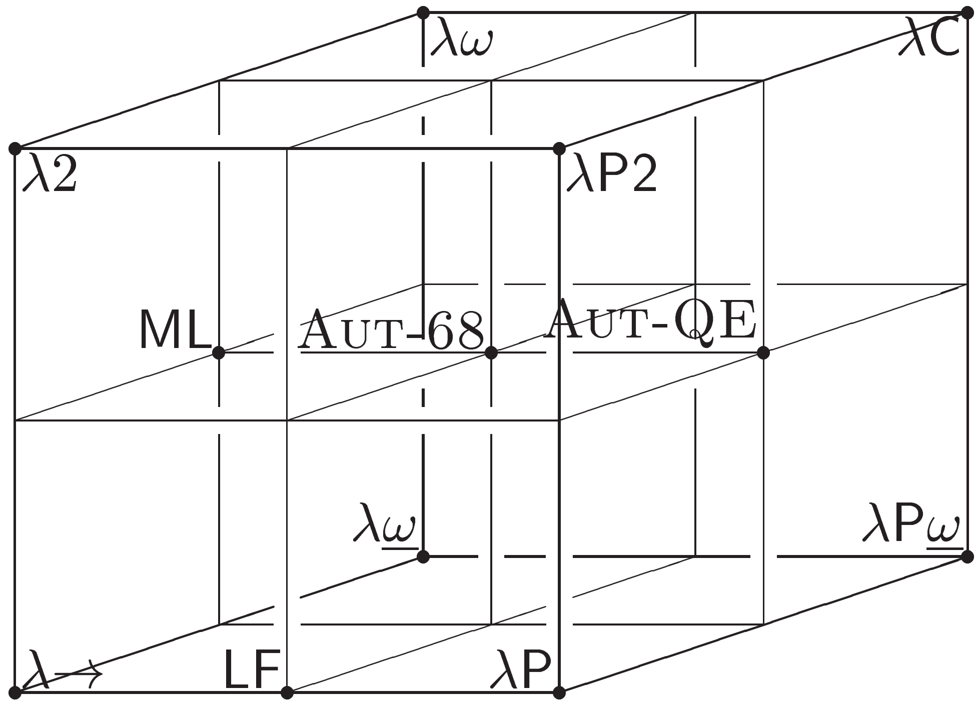

Figure 5: LF, ML, Aut-68, and Aut-QE in the refined Barendregt Cube

LF

- Geuvers [Geuvers, 1993] initially describes the system LF (see [Harper et al., 1987]) 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$^-$.

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 $\Gamma \vdash_{\mathsf{LF}} M : A$, and Γ, M, A do not contain a λ_P, then $\Gamma \vdash_{\mathsf{LF}^-} M : A$;
  - If $\Gamma \vdash M : A(: *)$, all in β_P-normal form, then $\Gamma \vdash_{\mathsf{LF}^-} M : A(: *)$.

- 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.

LF

- 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:

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

- 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 $p{:}\texttt{prop}$.

LF

- 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 $\lambda{\to}$ and λP in the Refined Barendregt Cube.

- The above only explained the extension of the Cube with parametric constants. Details can be found in [Kamareddine et al., 2001].

- 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

Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call by need lambda calculus. *22nd ACM Symposium on Principles of Programming Languages*, 1995.

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).

R. Bloo, F. Kamareddine, and R. P. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126 (2):123–143, 1996.

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.

P. de Groote. The conservation theorem revisited. In *International Conference on Typed Lambda Calculi and Applications, LNCS*, volume 664. Springer-Verlag, 1993.

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.

F. Kamareddine. A reduction relation for which postponement of K-contractions, Conservation and Preservation of Strong Normalisation hold. *Logic and Computation*, 10(5), 2000.

F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the barendregt cube using parameters. *Fifth International Symposium on Functional and Logic Programming, FLOPS 2001,*, Lecture Notes in Computer Science, 2001.

F. Kamareddine and R. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.

F. Kamareddine and R.P. Nederpelt. Generalising reduction in the λ-calculus. *Functional Programming*, 5 (4): 637–651, 1995.

F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *Lecture Notes in Computer Science*, 982:45–62, 1995.

F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Functional Programming*, 7(4):395–420, 1997.

F. Kamareddine and A. Ríos. Bridging the λs and $\lambda\sigma$ styles of Explicit Substitutions. *Logic and Computation*, 10 (3), 2000.

A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *ACM*, 41(2):368–398, 1994.

A.J. Kfoury and J.B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus. *Proceedings of the 1994 ACM Conference on LISP and Functional Programming*, 1994.

A.J. Kfoury and J.B. Wells. New notions of reductions and non-semantic proofs of β-strong normalisation in typed λ-calculi. *LICS*, 1995.

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

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. *Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types*. PhD thesis, Eindhoven University of Technology, 1973. Also in [Nederpelt et al., 1994], pages 389–468.

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.

L. Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.

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. All references are to the first volume, unless otherwise stated.