Lambda Calculus

Part III

Typing a la Church

Based on materials provided by H. P. Barendregt

We shall introduce in a uniform way the eight Lambda calculi typed a la Church

$$\lambda\to, \lambda 2, \lambda\underline{\omega}, \lambda\omega, \lambda P, \lambda P\underline{\omega}, \text{ and } \lambda P\omega.$$

The last one is often called λC the *calculus of constructions*. The eight systems form a cube as follows:

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda P\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda P2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda P\underline{\omega} \arrow[uu] \\
\lambda\to \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda P \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Each edge $\to$ represents the inclusion $\subseteq$. This cube will be referred to as the λ-cube.

As we have seen, the systems $\lambda\to$ and $\lambda 2$ can be given also *a la* Curry. A Curry version exists also for $\lambda\omega$ and something similar can be probably given for its weaker version $\lambda\underline{\omega}$.

On the other hand, no natural Curry versions of the systems $\lambda P, \lambda P2, \lambda P\underline{\omega}$ and λC seem possible.

Before we define the systems of the λ-cube in a uniform way, we introduce the systems $\lambda\to$ and $\lambda 2$ in the similar way as the Curry systems have been presented. Then it turns out that two of the systems of the λ-cube are equivalent to them.

Definition. $\lambda\to$ **– Church .**

•Types	$\mathbf{T} = \mathbf{V} \mid \mathbf{T} \to \mathbf{T}$
•Pseudoterms	$\Lambda_{\mathbf{T}} = V \mid \Lambda_{\mathbf{T}}\Lambda_{\mathbf{T}} \mid \lambda V : \mathbf{T}.\Lambda$
•Bases	$\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ with all x_i distinct and all $A_i \in \mathbf{T}$.
•Contraction Rule	$(\lambda x : A.M)N \to_\beta M[x := N]$
•Type assignment	$\Gamma \mid\!- M : A$ is defined as follows:

$\lambda\to$

(start rule) $$\frac{(x : A) \in \Gamma}{\Gamma \mid\!- x : A}$$

($\to$ –elimination) $$\frac{\Gamma \mid\!- M : (A \to B) \quad \Gamma \mid\!- N : A}{\Gamma \mid\!- (MN) : B}$$

($\to$ –introduction) $$\frac{\Gamma, x : A \mid\!- M : B}{\Gamma \mid\!- (\lambda x : A.M) : (A \to B)}$$

Where the basis $\Gamma, x : A$ stands for $\Gamma \cup \{x : A\}$ and it is necessary that the variable x does not occur in Γ. The letters A, B denote arbitrary types and M,N arbitrary pseudoterms.

Examples.

$$\mid\!- (\lambda a : A.a) : (A \to A)$$
$$b : B \mid\!- (\lambda a : A.b) : (A \to B)$$
$$b : A \mid\!- ((\lambda a : A.a)b) : A$$
$$c : A, b : B \mid\!- (\lambda a : A.b)c : B$$
$$\mid\!- (\lambda a : A.\lambda b : B.a) : (A \to B \to A)$$

The system $\lambda 2$-Church

•Types	$\mathbf{T} = \mathbf{V} \mid \mathbf{T} \to \mathbf{T} \mid \forall \mathbf{V}\mathbf{T}$
•Pseudoterms	$\Lambda_{\mathbf{T}} = V \mid \Lambda_{\mathbf{T}}\Lambda_{\mathbf{T}} \mid \Lambda_{\mathbf{T}}\mathbf{T} \mid \lambda V : \mathbf{T}\Lambda_{\mathbf{T}} \mid \Lambda \mathbf{V} \Lambda_{\mathbf{T}}$
•Bases	$\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ with all x_i distinct and $A_i \in \mathbf{T}$
•Contraction rules	$(\lambda a : A.M)N \to_\beta M[a := N]$ $(\Lambda\alpha.M)A \to_\beta M[\alpha := A]$
•Type assignment	$\Gamma \mid\!- M : A$ is defined as follows:

$\lambda 2$

(start rule)	$\dfrac{(x:A)\in\Gamma}{\Gamma \mid\!- x:A}$
$(\rightarrow -$elimination)	$\dfrac{\Gamma \mid\!- M:(A\rightarrow B) \quad \Gamma \mid\!- N:A}{\Gamma \mid\!- (MN):B}$
$(\rightarrow -$introduction)	$\dfrac{\Gamma, x:A \mid\!- M:B}{\Gamma \mid\!- (\lambda x:A.M):(A\rightarrow B)}$
$(\forall -$ elimination)	$\dfrac{\Gamma \mid\!- M:(\forall\alpha.A)}{\Gamma \mid\!- MB: A[\alpha := B]} \quad B\in \mathbf{T}$
$(\forall -$ introduction)	$\dfrac{\Gamma \mid\!- M:A}{\Gamma \mid\!- (\Lambda\alpha.M):(\forall\alpha.A)} \quad \alpha\notin FV(\Gamma)$

Examples.

$$\mid\!- (\lambda a:\alpha.a):(\alpha\rightarrow\alpha)$$
$$\mid\!- (\Lambda\alpha\lambda a:\alpha.a):(\forall\alpha.\alpha\rightarrow\alpha)$$
$$\mid\!- (\Lambda\alpha\lambda a:\alpha.a)A:(A\rightarrow A)$$
$$b:A \mid\!- (\Lambda\alpha\lambda a:\alpha.a)Ab:A$$

For more advanced,check that the following reduction holds

$$\mid\!- (\Lambda\alpha\lambda a:\alpha.a)Ab \rightarrow (\lambda a:A.a)b \rightarrow b$$
$$\mid\!- (\Lambda\beta\lambda a:(\forall\alpha.\alpha).a((\forall\alpha.\alpha)\rightarrow\beta)a):(\forall\beta.(\forall\alpha.\alpha)\rightarrow\beta)$$

For less advanced

$$\mid\!- (\Lambda\beta\lambda a:(\forall\alpha.\alpha).\alpha\beta):(\forall\beta.(\forall\alpha.\alpha)\rightarrow\beta)$$

Without a proof: Church-Rosser property holds for the reduction of pseudoterms in $\lambda 2$.

Dependency.

Types are dependent on terms and vice versa. There are four cases:

terms depending on terms
terms depending on types
types depending on terms
types depending on types

The first two sorts of dependency are presented in $\lambda\rightarrow$ and $\lambda 2$.

In $\lambda\rightarrow$, we have

$$F:A\rightarrow B \quad M:A \quad \Rightarrow \quad FM:B$$

Here FM is a term depending on a term, in particular on M.

In $\lambda 2$, we have

$$G:\forall\alpha.\alpha\rightarrow\alpha \quad A \text{ a type} \quad \Rightarrow \quad GA:A\rightarrow A$$

Hence for $G\equiv\Lambda\alpha\lambda a:\alpha.a$, we have GA a term depending on the type A.

In $\lambda\rightarrow$ and $\lambda 2$ one has also function abstraction for the two types of dependence. For the two examples above

$$\lambda m:A.Fm:A\rightarrow B$$
$$\Lambda\alpha.G\alpha:\forall\alpha.\alpha\rightarrow\alpha$$

The systems $\lambda\omega$ and λP.

We shall show the remaining two dependencies: in particular with types FA in $\lambda\omega$ depending on types and FM in λP depending on terms.

We will also have function abstractions for these dependencies both in $\lambda\omega$ and λP.

The system $\lambda\underline{\omega}$: Types depending on types.

$\alpha \to \alpha$ is a natural example of a type depending on a type α.

We would like to define a term $f = \lambda\alpha \in \mathbf{T}.\alpha \to \alpha$ with a new form of abstraction such that $f(\alpha) = \alpha \to \alpha$. This will be possible in $\lambda\underline{\omega}$. To do this, it is not possible to define types in an informal metalanguage as we have done so far. It is necessary generate the type by the system itself.

Kinds and constructors.

Informally, take a constant $*$ such that $\sigma : *$ corresponds to $\sigma \in \mathbf{T}$.
The informal statement

$$\alpha, \beta \in \mathbf{T} \Rightarrow (\alpha \to \beta) \in \mathbf{T}$$

Now be comes the formal

$$\alpha : *, \beta : * \mid\!- (\alpha \to \beta) : *$$

Now, we can write $f \equiv \lambda\alpha : *.\alpha \to \alpha$ for the f above. But we have to ask, where this f live. Neither on the level of terms, nor among the types.

It is necessary to introduce a new class K, the elements of which are called *kinds*.

$$\mathsf{K} = * \mid \mathsf{K} \to \mathsf{K}$$

Hence

$$*, * \to *, * \to * \to *, \ldots$$

are kinds and

$$\mathsf{K} = \{*, * \to *, * \to * \to *, \ldots\}$$

It is necessary to introduce one more class □ such that $k :$ □ corresponds to $k \in \mathsf{K}$. If $\mid\!- k :$ □ and $\mid\!- F : k$, then F is called a *constructor* of kind k. Each element of T will be a constructor of kind $*$.

Example.

We shall show later on that

$$\mid\!- \underbrace{(\lambda\alpha : *.\alpha \to \alpha)}_{f} : (* \to *)$$

Hence the above function f is a constructor of kind $* \to *$.

Although the types and terms can be kept separate, we will consider them as a subset of one general set $\mathcal{T}$ of pseudo-expressions.

Definition. Types and terms of $\lambda\underline{\omega}$:

•Sorts	$*$, □	two constants selected from C.
•Class	$\mathsf{K} = * \mid \mathsf{K} \to \mathsf{K}$	the elements are called *kinds.*

•A set of pseudoexpressions $\mathcal{T}$

$$\mathcal{T} = V \mid C \mid \mathcal{T}\mathcal{T} \mid \lambda V : \mathcal{T} \mid \mathcal{T} \to \mathcal{T}$$

where V is an infinite collection of variables and C of constants.

Statements, bases - a motivation

As terms and types belong to the same set T, the definition of statement is modified accordingly, bases have both types of variables as subjects and have to become linearly ordered. That is why we call them contexts.

The reason is that in $\lambda\omega$ one wants to derive

$$\alpha:*, x:\alpha \;|- x:\alpha$$
$$\alpha:* \;|- (\lambda x:\alpha.x):(\alpha\to\alpha)$$

But not

$$x:\alpha, \alpha:* \;|- x:\alpha$$
$$x:\alpha \;|- (\lambda\alpha:*.x):(*\to\alpha)$$

in which α occurs both free and bound.

Definition. Contexts for $\lambda\omega$.

(i) A *statement* of $\lambda\omega$ is of form $M:A$ with $M, A\in\mathrm{T}$.

(ii) A *context* is a finite linearly ordered set of statements with distinct variables as subjects. We shall denote them by $\Gamma,\Delta\ldots$

(iii) $<>$ denotes the empty context. If $\Gamma = <x_1:A_1,\ldots,x_n:A_n>$ then $\Gamma, y:B = <x_1:A_1,\ldots,x_n:A_n, y:B>$

(iv) The (type) *assignment* $\Gamma |-_{\lambda\omega} M:A$ is derived by the following axioms and rules. The letter s ranges over sorts.

$\lambda\omega$

(axiom)	$<>|- *:\square$	
(start rule)	$\dfrac{\Gamma|- A:s}{\Gamma, x:A|- x:A}$	$x\notin\Gamma$
(weakening rule)	$\dfrac{\Gamma|- A:B \quad \Gamma|- C:s}{\Gamma, x:C|- A:B}$	$x\notin\Gamma$
(type/kind formation)	$\dfrac{\Gamma|- A:s \quad \Gamma|- B:s}{\Gamma|- (A\to B):s}$	
(application rule)	$\dfrac{\Gamma|- F:(A\to B) \quad \Gamma|- a:A}{\Gamma|- Fa:B}$	
(abstraction rule)	$\dfrac{\Gamma, x:A|- b:B \quad \Gamma|- (A\to B):s}{\Gamma|- (\lambda x:A.b):(A\to B)}$	
(conversion rule)	$\dfrac{\Gamma|- A:B \quad \Gamma|- B':s \quad B=_\beta B'}{\Gamma|- A:B'}$	

Examples.

$$\alpha:*, \beta:* |-_{\lambda\omega} \alpha\to\beta:*$$
$$\alpha:*, \beta:*, x:(\alpha\to\beta) |-_{\lambda\omega} x:(\alpha\to\beta)$$
$$\alpha:*, \beta:* |-_{\lambda\omega} (\lambda x:(\alpha\to\beta).x):((\alpha\to\beta)\to(\alpha\to\beta))$$

Put $D\equiv\lambda\beta:*.\beta\to\beta.$ Then the following hold.

$$|-_{\lambda\omega} D:(*\to *)$$
$$\alpha:* |-_{\lambda\omega} (\lambda x:D\alpha.x):D(D\alpha)$$

The system λP: Types depending on terms.

$A^n \to B$ is an intuitive example of a type depending on a term. In order to formalize this dependence in λP, we need to extend the class **K** of kinds as follows:

if A is a type and $k \in \mathbf{K}$ then $A \to k \in \mathbf{K}$.

In particular $A \to *$ is a kind and if $f : A \to *$ and $a \in A$, one has $fa : *$.

The expression fa is a type depending on a term. Moreover, we have function abstraction for this dependency.

Cartesian products.

Suppose that for each $a{:}A$ a type B_a is given such that there is an element $b_a : B_a$. Then we may want to form the function

$$\lambda a : A.b_a$$

that should have as a type the cartesian product

$$\prod a : A.B_a$$

of types B_a´s.

Once these product types are allowed, the type constructor $\to$ can be eliminated. We can write

$$(A \to B) \equiv \prod a : A.B$$

where a is a variable not occuring in B.

Types and terms of λP.

(i) The set T of pseudo-expressions of λP is defined as follows

$$\mathrm{T} \equiv V \mid C \mid \mathrm{TT} \mid \lambda V : \mathrm{T}.\mathrm{T} \mid \Pi V : \mathrm{T}.\mathrm{T}$$

Where V is the set of all variables and C that of constants. No distinction between type variables and term variables is made.

(ii) Among the constants C two elements are called $*$ and $\square$.

Assignment rules for λP.

Statements of the form $M : A$ with $M, A \in \mathrm{T}$ and contexts are defined as for $\lambda\omega$.

Contexts are finite linearly ordered sequences of statements.

Sorts are two constants denoted by $*$ and $\square$. Again, the letter s ranges over the set of sorts.

The notion $\vdash$ is defined by the following axiom and rules.

λP

(axiom)
$$<>\,|-*:\square$$

(start-rule)
$$\frac{\Gamma\,|-A:s}{\Gamma,x:A\,|-x:A}\quad x\notin\Gamma$$

(weakening rule)
$$\frac{\Gamma\,|-A:B\quad \Gamma\,|-C:s}{\Gamma,x:C\,|-A:B}\quad x\notin\Gamma$$

(type/kind formation)
$$\frac{\Gamma\,|-A:*\quad \Gamma,x:A\,|-B:s}{\Gamma\,|-(\Pi x:A.B):s}$$

(application rule)
$$\frac{\Gamma\,|-F:(\Pi x:A.B)\quad \Gamma\,|-a:A}{\Gamma\,|-Fa:B[x:=a]}$$

(abstraction rule)
$$\frac{\Gamma,x:A\,|-b:B\quad \Gamma\,|-(\Pi x:A.B):s}{\Gamma\,|-(\lambda x:A.b):(\Pi x:A.B)}$$

(conversion rule)
$$\frac{\Gamma\,|-A:B\quad \Gamma\,|-B':s\quad B=_\beta B'}{\Gamma\,|-A:B'}$$

Exercises.

$$A:*\,|-(A\to *):\square$$
$$A:*,\ P:A\to *,a:A\,|-Pa:*$$
$$A:*,\ P:A\to *,a:A\,|-Pa\to *:\square$$
$$A:*,P:A\to *\,|-(\Pi a:A.Pa\to *):\square$$
$$A:*,P:A\to *\,|-(\lambda a:A\,\lambda x:Pa.x):(\Pi a:A.(Pa\to Pa))$$

λP and Logic. (Pragmatics of λP)

Systems similar to λP have been introduced by
N. G. De Bruijn in the 1970s and 1980s in order to represent
mathematical theorems and their proofs.

Idea. Assume that there is a set *prop* closed under implication.
This can be done by context

$$\Gamma_0\equiv <prop:*,Imp:prop\to prop\to prop>$$

•We shall write $\varphi\supset\psi$ for $Imp\,\varphi\psi$.

•A variable $T:prop\to *$ is declared and $\varphi:prop$
is declared to be valid if $T\varphi$ is inhabited.

To guarantee that the implication has the right properties, one
assumes $\supset_e$ and $\supset_i$ such that

$$\supset_e\varphi\psi:T(\varphi\supset\psi)\to T\varphi\to T\psi$$
$$\supset_i\varphi\psi:(T\varphi\to T\psi)\to T(\varphi\supset\psi)$$

Now for representation of implicational proposition logic
we choose to work in context Γ_{prop} consisting of

$$\Gamma_0$$
$$T:prop\to *$$
$$\supset_e:\Pi\varphi:prop\Pi\psi:prop.T(\varphi\supset\psi)\to T\varphi\to T\psi$$
$$\supset_i:\Pi\varphi:prop\Pi\psi:prop.(T\varphi\to T\psi)\to T(\varphi\supset\psi)$$

Example.

We want to show that $\varphi \supset \varphi$ is valid for all propositions. We need to show that its translation as a type $T(\varphi \supset \varphi)$ is inhabited.

We have

$$
\begin{array}{ll}
\dfrac{\dfrac{\dfrac{\varphi : prop \quad T : prop \to *}{T\varphi : *}\qquad \dfrac{}{\cancel{x : T\varphi}}\ 1}{(\lambda x : T\varphi.x) : (T\varphi \to T\varphi)}\ 1}{(\supset_i \varphi\varphi(\lambda x : T\varphi.x)) : T(\varphi \to \varphi)} &
\begin{array}{l}\text{(context)}\\ \text{(application)}\\ \text{(assumption)}\\ \text{(abstraction)}\\ \text{(context, application)}\end{array}
\end{array}
$$

Hence

$$\Gamma_{prop} \mid\!-_{\lambda P} (\supset_i \varphi\varphi(\lambda x : T\varphi.x)) : T(\varphi \to \varphi)$$

Simplified notation.

$$
\begin{array}{l}
prop \cdots\cdots *\\
\supset \cdots\cdots\cdots \to\\
T \cdots\cdots\cdots \mathbf{I} \quad \text{(identity)}
\end{array}
$$

Then for $\supset_e \varphi\psi$ one can use

$$\lambda x : (\varphi \to \psi)\lambda y : \varphi.xy$$

and for $\supset_i \varphi\psi$

$$\lambda x : (\varphi \to \psi).x$$

In this way the $\{\to, \forall\}$ fragment of (manysorted, constructive) predicate logic can be interpreted in λP. A predicate on a type with the domain A is represented as the statement $P : (A \to *)$.

One defines Pa for $a{:}A$ to be valid, if it is inhabited.

Quantification is translated as follows

$$\forall x \in A.Px \quad \cdots\cdots> \quad \Pi x : A.Px$$

Example.

Formula

$$(\forall x \in A \forall y \in A.Pxy) \to (\forall x \in A.Pxx)$$

is valid, since its translation is inhabited:

$$
\begin{array}{ll}
A : *, P : A \to A \to * & \mid- (\lambda z : (\Pi x : A\Pi y : A.Pxy)\lambda x : A.zxx) :\\
& \quad ([\Pi x : A\Pi Y : A.Pxy] \to [\Pi x : A.Pxx])
\end{array}
$$

The system λP deserved its name because predicate logic can be interpreted in it.

The method interprets

propositions, formulas	$\cdots\cdots$	types
proofs	$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$	terms inhabiting the types

The method is often called *propositions as types paradigm* and it is used for formulating results in the foundations of mathematics.

Systems of the λ-cube: A uniform definition.

(i) The set T of pseudo-expressions of λP is defined as follows

$$\mathrm{T} \equiv V \mid C \mid \mathrm{TT} \mid \lambda V:\mathrm{T.T} \mid \Pi\, V:\mathrm{T.T}$$

Where V is the set of all variables and C that of constants. No distinction between type variables and term variables is made.

We use A,B,C,…a,b,c,… for pseudo-terms and x,y,z,… for variables.

(ii) Two constants are selected and denoted by $*$ and □. They are called *sorts*. The letter s ranges over the set of sorts.

(iii) On T the notions of β-conversions and β-reductions are defined by the following contraction rule

$$(\lambda x:A.B)C \to B[x:=C]$$

(iv) A statement is of the form $A:B$ where $A,B \in \mathrm{T}$.
We call A the *subject* and B the *predicate* of the statement *A:B*.

A declaration is a statement with a variable as the subject and with a pseudo-expression as the predicate.

(v) A *pseudo-context* is a finite ordered sequence of declarations, all with distinct subjects. The empty pseudo-context is denoted by <> (usually we do not write it).

Given a pseudo-context $\Gamma =< x_1 : A_1,\dots, x_n : A_n >$

its extension is defined as follows:

$$\Gamma, x:B = < x_1 : A_1,\dots, x_n : A_n, x:B > .$$

(vi) The notion $\Gamma \mid - A:B$ states that *A:B* can be derived from the pseudo-context Γ, in this case we say that A and B are legal expressions and Γ is a legal pseudo-context. The notion is axiomatized by the rules of type assignment.

The rules are divided into two groups:

a) general axiom and rules valid for all systems of λ-cube,

b) the specific rules differentiating the eight systems (usually parametrized Π-introduction rules).

Systems of the λ-cube

1. General axiom and rules.

(axiom) $$<>\mid - *:□$$

(start-rule) $$\frac{\Gamma \mid - A:s}{\Gamma, x:A \mid - x:A} \quad x \notin \Gamma$$

(weakening rule) $$\frac{\Gamma \mid - A:B \quad \Gamma \mid - C:s}{\Gamma, x:C \mid - A:B} \quad x \notin \Gamma$$

(application rule) $$\frac{\Gamma \mid - F:(\Pi x:A.B) \quad \Gamma \mid - a:A}{\Gamma \mid - Fa:B[x:=a]}$$

(abstraction rule) $$\frac{\Gamma, x:A \mid - b:B \quad \Gamma \mid - (\Pi x:A.B):s}{\Gamma \mid - (\lambda x:A.b):(\Pi x:A.B)}$$

(conversion rule) $$\frac{\Gamma \mid - A:B \quad \Gamma \mid - B':s \quad B =_\beta B'}{\Gamma \mid - A:B'}$$

2. The specific rules.

(s_1, s_2) rule $\qquad \dfrac{\Gamma \mid\!- A : s_1 \qquad \Gamma, x : A \mid\!- B : s_2}{\Gamma \mid\!- (\Pi x : A.B) : s_2}$

We select four specific rules

$$(*, *), (*, □), (□, *), (□, □)$$

And the eight systems of λ-cube consist of the general rules together with a specific subsets of the above specific rules.

The sets of specific rules for the eight systems are depicted in the table below.

System	Set of specific rules			
$\lambda\to$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(□,*)$		
λP	$(*,*)$		$(*,□)$	
$\lambda P2$	$(*,*)$	$(□,*)$	$(*,□)$	
$\lambda\underline{\omega}$	$(*,*)$			$(□,□)$
$\lambda\omega$	$(*,*)$	$(□,*)$		$(□,□)$
$\lambda P\underline{\omega}$	$(*,*)$		$(□,*)$	$(□,□)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(□,*)$	$(*,□)$	$(□,□)$

λ-cube

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda P\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda P2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda P\underline{\omega} \arrow[uu] \\
\lambda\to \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda P \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

System	related system	names and references
$\lambda\to$	λ^{τ}	Simply typed lambda calculus [Church 1940] [Barendregt 1984] [Hindley and Seldin 1986]
$\lambda 2$	F	Second order (typed) lambda calculus [Girard 1972] [Reynolds 1974]
λP	$AUT-QE, LF$	[de Bruin 1970] [Harper et al. 1987]
$\lambda P2$		[Longo and Moggi 1988]
$\lambda\underline{\omega}$	$POLYREC$	[Renardel de Lavalette 1991]
$\lambda\omega$	$F\omega$	[Girard 1972]
$\lambda P\omega$	CC	Calculus of constructions [Coquand and Huet]

Remarks.

(i) Impredicativity. The expression

$$(\Pi\alpha:*.(\alpha\to\alpha))$$

in $\lambda2$ as a cartesian product of types, will be a type, too. So

$$|-(\Pi\alpha:*.(\alpha\to\alpha)):*$$

but since it is a product over all possible types α,
including $(\Pi\alpha:*.(\alpha\to\alpha))$ itself,
there is an essential impredicativity here.

(ii) Terms depending on types and types depending on types in $\lambda\to$.

$\lambda x:A.x$ is a term depending on type A
$A\to A$ is a type depending on the type A

but in $\lambda\to$ we have no function abstraction for these dependencies.

(iii) Note that in $\lambda\to$ and even in $\lambda 2$ and $\lambda\omega$ one has no types depending on terms. The types are given beforehand. Thus the right-hand side of the cube is essentially more difficult than the left-hand side because of the mixture of types and terms.

Equivalence of both versions of $\lambda\to$ and $\lambda 2$.

Recall the definition $A\to B\equiv\Pi x:A.B$ where x is not in A, B.

Notice that application rule in the λ-cube implies the $(\to-$elimination$)$ rule:

$$\frac{\Gamma|-F:(A\to B)\equiv\Pi x:A.B \qquad \Gamma|-a:A}{\Gamma|-(Fa):B[x:=a]\equiv B}$$

Since x does not occur in B. It follows that if we have

$$A:*,B:*,a:A,b:B|-M:C:*$$

in $\lambda\to$ in λ-cube then

$$a:A,b:B|-M:C$$

is derivable in the original system $\lambda\to$. The notation $\Gamma|-M:C:*$ stands for the conjunction $\Gamma|-M:C$ and $\Gamma|-C:*$.

Lemma.

Consider $\lambda\to$ in the λ-cube. If $\Gamma|-A:*$ in this system, then A is built up from the set $\{B\,|\,(B:*)\in\Gamma\}$ using only $\to$ as defined above.

Proof.

By induction on the generation of $|-$.

To show that both versions of $\lambda2$ are the same, we have to define

$$\forall\alpha.A\equiv\Pi\alpha:*.A$$
$$\Lambda\alpha.M\equiv\lambda\alpha:*.M$$

in the λ-cube. .

Examples.

(a) in $\lambda\to$ one can derive

$$\begin{array}{rl}
A:* & \vdash (\Pi x:A.A):* \\
A:* & \vdash (\lambda a:A.a):(\Pi x:A.A) \\
A:*,B:*,b:B & \vdash (\lambda a:A.b):(A\to B) \\
& \quad \text{where } (A\to B)\equiv(\Pi x:A.B)
\end{array}$$

$$\begin{array}{rl}
A:*,b:A & \vdash ((\lambda a:A.a)b):A \\
A:*,B:*,c:A,b:B & \vdash ((\lambda a:A.a)b):A \\
A:*,B:* & \vdash (\lambda a:A\lambda b:B.a):(A\to(B\to A)):*
\end{array}$$

(b) In $\lambda 2$ one can derive

$$\begin{array}{rl}
\alpha:* & \vdash (\lambda a:\alpha.a):(\alpha\to\alpha) \\
& \vdash (\lambda\alpha:*\lambda a:\alpha.a):(\Pi\alpha:*.(\alpha\to\alpha)):* \\
A:* & \vdash (\lambda\alpha:*\lambda a:\alpha.a)A:(A\to A) \\
A:*,b:A & \vdash (\lambda\alpha:*\lambda a:\alpha.a)Ab:A
\end{array}$$

Notice that for the last line the following reduction holds:

$$\begin{array}{rcl}
(\lambda\alpha:*\lambda a:\alpha.a)Ab & \to & (\lambda a:A.a)b \\
& \to & b
\end{array}$$

Connection of $\lambda 2$ with second-order propositional logic.

Exercise.

$$\vdash_{\lambda 2} \underbrace{(\lambda\beta:*\lambda a:(\Pi\alpha:*.a).a((\Pi\alpha:*.\alpha)\to\beta)a)}_{subject}:\underbrace{(\Pi\beta:*.(\Pi\alpha:*.\alpha)\to\beta)}_{predicate}$$

Simplification: put $\bot\equiv(\Pi\alpha:*.\alpha)$ which is the definition of the second-order *falsum*. Using this, we may write

$$\vdash \underbrace{(\lambda\beta:*\lambda a:\bot.a\beta)}_{subject}:\underbrace{(\Pi\beta:*.\bot\to\beta)}_{predicate}$$

The predicate (type) considered as a proposition says: *ex falso sequitur quodlibet* („anything follows from a false statement“) and the subject (term) is its proof.

(c) In $\lambda\omega$ one can derive e.g.

$$\vdash (\lambda\alpha:*.\alpha\to\alpha):(*\to *):\Box$$

{$(\lambda\alpha:*.\alpha\to\alpha)$ is a constructor mapping types into types}

Proof.

$$(*,*)\quad \dfrac{\dfrac{\alpha:* \qquad\qquad x:\alpha}{\underbrace{(\Pi x:\alpha.\alpha)}_{\alpha\to\alpha}:*} \qquad\qquad \dfrac{*:\Box \quad x:* \quad *:\Box}{\underbrace{(\Pi x:*.*)}_{*\to *}:\Box}}{(\lambda\alpha:*.\alpha\to\alpha):(*\to *):\Box}\qquad (\Box,\Box)$$

Similarly one can derive

$$\beta:*\vdash(\lambda\alpha:*.\alpha\to\alpha)\beta:*$$
$$\beta:*,x:\beta\vdash(\lambda y:\beta.x):(\lambda\alpha:*.\alpha\to\alpha)\beta$$

Higher-order constructors.

They are formed in the following way

$$\alpha:*,\ f:*\to*\ |-\ f(f\alpha):*$$

$$\alpha:*\ |-\ (\lambda f:*\to*.f(fa)):(*\to*)\to*$$

(d) λP-Propositions as types.

the following can be derived:

$$A:*\ |-\ (A\to*):\Box$$

{If A is a type considered as a set, then $A\to*$ is the kind of predicates on A.}

(i) if A is a non-empty set, $a\in A$ and P is a predicate on A, then Pa is a type considered as a proposition which is true if inhabited, otherwise false.

$$A:*,\ P:(A\to*),\ a:A\ |-\ Pa:*$$

(ii) if P is a binary predicate on the set A, then $\forall a\in A\ Paa$ is a proposition.

$$A:*,\ P:(A\to A\to*)\ |-\ (\Pi a:A.Paa):*$$

(iii) if P and Q are two unary predicates on a set A, then the predicate P considered as a set is included in Q.

$$A:*,\ P:A\to*,\ Q:A\to*\ |-\ (\Pi a:A.(Pa\to Qa)):*$$

(iv) proposition stating the reflexivity of inclusion.

$$A:*,\ P:A\to*\ |-(\Pi a:A(Pa\to Pa)):*$$

(v) „proof“ of the reflexivity of inclusion.

$$A:*,\ P:A\to*\ |-\ \underbrace{(\lambda a:A\,\lambda x:Pa.x)}_{subject}:(\Pi a:A.(Pa\to Pa)):*$$

(vi) A type considered as a (true) proposition.

$$A:*,\ P:A\to*,\ Q:*\ |-\ ((\Pi a:A.Pa\to Q)\to(\Pi a:A.Pa)\to Q):*$$

We have proved that the type on the right hand is a proposition, to be true, we have to assume that A is non-empty.

$$A:*,\ P:A\to*,\ Q:*,\ a_0:A\ |-$$
$$(\lambda x:(\Pi a:A.Pa\to Q)\ \lambda y:(\Pi a:A.Pa).xa_0(ya_0)Q):$$
$$\underbrace{(\Pi x:(\Pi a:A.Pa\to Q)\Pi y:(\Pi a:A.Pa).Q)}_{(\Pi a:A.Pa\to Q)\to(\Pi a:A.Pa)\to Q}$$

This proposition as a type states that proposition

$$(\forall a\in A.Pa\to Q)\to(\forall a\in A.Pa)\to Q$$

is true in non-empty structures A.

(e) λω - conjunction

The second-order definition of conjuntion is defined as follows

$$\alpha \,\&\, \beta \equiv \Pi\gamma:*.(\alpha \to \beta \to \gamma) \to \gamma,$$

{it is definable already in λ2}, but in λω can be derived

$$\alpha:*,\beta:* \mid\!- \alpha \,\&\, \beta : *$$

Let

$$AND \equiv \lambda\alpha:*\,\lambda\beta:*.\alpha \,\&\, \beta \quad \text{and} \quad K \equiv \lambda\alpha:*\,\lambda\beta:*\,\lambda x:\alpha\,\lambda y:\beta.x$$

then

$$\mid\!- AND : (* \to * \to *)$$
$$\mid\!- K : (\Pi\alpha:*\,\Pi\beta:*.\alpha \to \beta \to \alpha)$$

Note that while $\alpha\&\beta$ and K can be derived already in λ2, AND cannot.

The subject of the following assignment is a proof that

$$AND\alpha\beta \to \alpha$$

is a tautology.

$$\alpha:*,\beta:* \mid\!- (\lambda x:AND\alpha\beta.x\alpha(K\alpha\beta)) : (AND\alpha\beta \to \alpha) : *$$

(f) λP2 is corresponding to second-order predicate logic.

In it the following can be derived

$$A:*,\, P:A \to * \mid\!- (\lambda a:A.\, Pa \to \bot) : (A \to *)$$
$$A:*,\, P:A \to A \to * \mid\!- [(\Pi a:A\,\Pi b:A.Pab \to Pba \to \bot) \to$$
$$\to (\Pi a:A.Paa \to \bot)] : *$$

The proposition states that a binary relation that is asymetric is irreflexive.

(g) λP$\underline{\omega}$ gives the following derivation.

$$A:* \mid\!- (\lambda P:A \to A \to *\,\lambda a:A.Paa) : ((A \to A \to *) \to (A \to *)) :$$

This constructor assigns to a binary predicate P on A its diagonalization. The same can be done uniformly in A.

$$\mid\!- (\lambda A:*\,\lambda P:A \to A \to *\,\lambda a:A.Paa) :$$
$$(\Pi A:*\,\Pi P:A \to A \to *\,\Pi a:A.*) :$$

(h) λPω = λC The calculus of constructions.

(i) A constructor can be derived that assigns to a type A and to a predicate P on A the negation of P.

$$|-(\lambda A:*\,\lambda P:A\to *\,\lambda a:A\,.Pa\to\bot):$$
$$(\Pi A:*\,.(A\to *)\to(A\to *)):$$

(ii) Universal quantification done uniformly:

Let $ALL \equiv (\lambda A:*\,\lambda P:A\to *\,.\Pi a:A\,.Pa)$ then

$A:*, P:A\to * \mid- ALL\ AP:*$ and $(ALL\ AP) =_\beta (\Pi a:A\,.Pa)$

Exercises.

a) Define $\neg \equiv \lambda\alpha:*\,.\alpha\to\bot$. Construct a term M such that in λω

$$\alpha:*, \beta:* \mid- M:((\alpha\to\beta)\to(\neg\beta\to\neg\alpha))$$

b) Find an expression M such that in λP2, we have

$$A:*, P:(A\to A\to *) \mid-$$
$$M:[(\Pi a:A\,\Pi b:A\,.Pab\to Pba\to\bot)\to(\Pi a:A\ Paa\to\bot)]:*$$

c) Find a term M such that in λC, we have

$$A:*, P:A\to *, a:A \mid- M:(ALL\ AP\to Pa)$$

Pure Type systems: A generalization of the λ-cube.

• Many systems of typed lambda calculus *a la* Church can be seen as Pure Type Systems.

• One of the successes of the notion of Pure Type Systems is concerned with Logic: eight logical systems are shown to be in correspondence with the systems on the λ-cube.

• The general setting of Pure Type systems makes it easier to give the required proof

The pure types systems are based on the same set of pseudoterms as systems of the λ-cube.

$$\mathrm{T} = V \mid C \mid \mathrm{TT} \mid \lambda V:\mathrm{TT} \mid \Pi V:\mathrm{TT}$$

Definition. The *specification* of a Pure type system consists of a triple S = (S, A, R) where

• *S* is a subset of C, the elements of *S* are called *sorts.*

• *A* is a set of axioms of the form

c : s

with $c\in C$ and $s\in S$.

R is a set of rules of the form

(s_1, s_2, s_3) with $s_1, s_2, s_3 \in S$.

Thw set of variables V is stratified according to sorts into disjoint infinite subsets V_s for each sort $s \in S$. Hence $V = \cup\{V_s \mid s \in S\}$. The members of V_s are denoted ${}^s x,\ {}^s y,\ {}^s z, \ldots$

Arbitrary variables are still denoted by $x, y, z, \ldots$ if necessary one writes $x \equiv {}^s x$ for $x \in V_s$.

The first version of $\lambda 2$ can be understood as

$x, y, z, \ldots$ ranging over V_*

and

$\alpha, \beta, \gamma, \ldots$ over V

Definition.

The pure type system given by specification $S = (S, A, R)$ is denoted by $\lambda S = \lambda(S, A, R)$. Its properties are defined as follows.

•Statements and contexts are defined as for the λ-cube

•The notion of type derivations $\Gamma \vdash_{\lambda S} A : B$ is defined by the following axioms and rules

$\lambda(S, A, R)$

(axioms)	$<> \vdash c : s$	if $(c : s) \in A$
(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	if $x \equiv {}^s x \notin \Gamma$
(weakening)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$	if $x \equiv {}^s x \notin \Gamma$
(product)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_3}$	$(s_1, s_2, s_3) \in R$
(application)	$\dfrac{\Gamma \vdash F : (\Pi x : A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$	
(abstraction)	$\dfrac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash (\lambda x : A.b) : (\Pi x : A.B)}$	
(conversion)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$	

The side condition $(B =_\beta B')$ is not decidable. However it can be replaced by the decidable condition

$$B' \twoheadrightarrow B \quad \text{or} \quad B \twoheadrightarrow B'$$

with no effect on the set of derivable statements.

Definition.

(i) The rule (s_1, s_2) is an abbreviation for (s_1, s_2, s_2). In the λ-cube only systems with rules of this simple form are used.

(ii) The Pure type system is *full* if

$$R = S \times S = \{(s_1, s_2) \mid s_1, s_2 \in S\}.$$

Examples.

$\lambda\to$

S	$*, □$
A	$*: □$
R	$(*,*)$

$\lambda 2$

S	$*, □$
A	$*: □$
R	$(*,*), (□,*)$

λC

S	$*, □$
A	$*: □$
R	$(*,*), (□,*), (*,□), (□,□)$

It is a full system.

The system of higher order logic [Church 1940] can be described as follows.

λHOL

S	$*, □, \Delta$
A	$*: □,\ □: \Delta$
R	$(*,*), (□,*), (□,□)$

The system below is a subsystem of $\lambda\to$. An interesting conjecture of de Bruijn states that mathematics from before the year 1800 can all be formalized in it.

λPAL

S	$*, □, \Delta$
A	$*: □$
R	$(□, □, \Delta), (*, \Delta, \Delta)\ (□, \Delta, \Delta)$

Definition. Legal contexts and legal pseudoterms.

Let Γ be a pseudocontext and A a pseudoterm.

(i) Γ is called *legal* if Γ |- P:Q for some pseudoterms P,Q.
(ii) A pseudoterm A is called *legal* if there is a pseudocontext Γ and pseudoterm B such that Γ|-A:B or Γ|-B:A.

Transitivity lemma.

Let Γ and Δ be contexts of which Γ is legal. Then

$$[\Gamma \mid\!- \Delta \text{ and } \Delta \mid\!- A:B] \Rightarrow \Gamma \mid\!- A:B$$

Substitution lemma.

Assume $\Gamma, x: A, \Delta \mid\!- B : C$

and $\Gamma \mid\!- D : A$

Then $\Gamma, \Delta[x := D] \quad \mid\!- B[x := D] : C[x := D]$

Thinning lemma.

Let Γ and Δ be legal contexts and $\Gamma \subseteq \Delta$. Then

$$\Gamma \mid\!- A : B \Rightarrow \Delta \mid\!- A : B$$

Generation lemma

(*i*) $\Gamma \vdash c : C \quad \Rightarrow \quad \exists s \in S\ [C =_\beta s \;\&\; (c:s) \in A]$

(ii) $\Gamma \vdash x : C \quad \Rightarrow \quad \exists s \in S\ \exists B =_\beta C\ [\Gamma \vdash B : s \;\&\; (x:B) \in \Gamma$
$\& \; x \equiv^s x]$

(iii) $\Gamma \vdash (\Pi x : A.B) : C \quad \Rightarrow \quad \exists (s_1, s_2, s_3) \in R\ [\Gamma \vdash A : s_1 \;\&$
$\Gamma, x : A \vdash B : s_2 \;\&\; C =_\beta s_3]$

(iv) $\Gamma \vdash (\lambda x : A.b) : C \quad \Rightarrow \quad \exists s \in S\ \exists B\ [\Gamma \vdash (\Pi x : A.B) : s \;\&$
$\Gamma, x : A \vdash b : B \;\&\; (C =_\beta (\Pi x : A.B))]$

(v) $\Gamma \vdash (Fa) : C \quad \Rightarrow \quad \exists A, B\ [\Gamma \vdash F : (\Pi x : A.B) \;\&$
$\Gamma \vdash a : A \;\&\; C =_\beta B[x := a]]$

Subject reduction theorem.

$$\Gamma \vdash A : B \;\&\; A \twoheadrightarrow_\beta A' \quad \Rightarrow \quad \Gamma \vdash A' : B$$

Condensing lemma.

If x is not free in Δ, B, C, then

$$\Gamma, x : A, \Delta \vdash B : C \Rightarrow \Gamma, \Delta \vdash B : C$$

Definition. Simply sorted systems.

Let λ*S* = λ(*S*,*A*,*R*) be a Pure type system. λ*S* is called *singly sorted* if

(i) $(c_1 : s_1), (c_2 : s_2) \in A \quad \Rightarrow \quad s_1 \equiv s_2$

(ii) $(s_1, s_2, s_3), (s_1, s_2, s_3') \in R \quad \Rightarrow \quad s_3 \equiv s_3'$

Uniqueness of types for singly sorted Pure type systems.

Let λ*S* be a singly sorted Pure type system. Then

$$\Gamma \vdash A : B_1 \;\&\; \Gamma \vdash A : B_2 \quad \Rightarrow \quad B_1 =_\beta B_2$$

Definition. Strong normalization for the λ-cube.

Let λ*S* be a Pure type system. We call it *strongly normalizing* and write λS |= *SN* if all legal terms of λ*S* are SN, i.e

$$\Gamma \vdash A : B \quad \Rightarrow \quad SN(A) \;\&\; SN(B)$$

Theorem. Strong normalization for the λ-cube.

For all systems in the λ-cube, we have the following

(i) $\Gamma \vdash A : B \quad \Rightarrow \quad SN(A) \;\&\; SN(B)$

(ii) $x_1 : A_1, \ldots, x_n : A_n \vdash B : C \Rightarrow A_1, \ldots, A_n, B, C$ are SN

Representing logics.

Eight systems of intuitionistic logic correspond in some sense to the systems in the λ-cube: there are four systems of propositional logic and four systems of many sorted predicate logic.

PROP	propositional logic
PROP2	second-order propositional logic
PROP$\underline{\omega}$	weakly higher-order proposition logic
PROPω	higher-order proposition logic
PRED	predicate logic
PRED2	second-order predicate logic
PRED$\underline{\omega}$	weakly higher-ordered predicate logic
PREDω	higher-order predicate logic

•All these systems are minimal logics, the only logical operators are $\rightarrow$ and $\forall$.

• However, for the second- and higher-order systems, the operators $\neg, \&, \vee$ and $\exists$ are all definable.

• Weakly higher-order logics have variables for higher-order propositions or predicates but no quantification over them.

• A higher-order propositions have lower order propositions as arguments.

• All the above logics are intuitionistic. The classical versions of the logics in the upper plane of the *logic-cube* (see below) are obtained by adding as axiom $\forall\alpha.\neg\neg\alpha \rightarrow \alpha$.

• The systems form a cube as shown below.

$$\begin{tikzcd}
 & PROP_{\omega} \arrow[rr] & & PRED\omega \\
PROP2 \arrow[ur] \arrow[rr] & & PRED2 \arrow[ur] & \\
 & PROP\underline{\omega} \arrow[uu] \arrow[rr] & & PRED\underline{\omega} \arrow[uu] \\
PROP \arrow[uu] \arrow[ur] \arrow[rr] & & PRED \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

This cube will be referred as logic-cube.

Each system L_i on the logic-cube corresponds to the system λ_i on the corresponding vertex. The edges of the logic-cube represent inclusions of the systems in the same way as on the λ-cube.

Propositions as types: the idea.

A formula in the logic L_i on the logic-cube can be interpreted as a type $\|A\|$ in the corresponding λ_i on the λ-cube .

The transition

$$A \mapsto \|A\|$$

Is called *propositions-as-types* interpretation of L_i.

Soundness.

The propositions-as-types interpretation satisfies the following soundness result:

If A is provable in PRED, then $\|A\|$ is inhabited in λP

In fact an inhabitant of $\|A\|$ in λP can be found canonically from a proof of A in PRED. Different proofs of A are interpreted as different terms of type $\|A\|$.

Soundness can be shown all systems L_i with respect to the corresponding systems λ_i of the λ-cube .

Completness.

Completness is defined naturally: if A is a formula of the logic L_i such that the type $\|A\|$ is inhabited in λ_i,then A is provable in L_i.

For the proposition logics it is trivially true. Completnes was proved for PRED with respect to λP. For PREDω with respect to λC fails.