Open access • Journal Article • DOI:10.1017/S0956796800020025

Introduction to generalized type systems — Source link

Henk Barendregt

Institutions: The Catholic University of America

Published on: 01 Apr 1991 - Journal of Functional Programming (Cambridge University Press (CUP))

Topics: Typed lambda calculus and Canonical form

Journal of Functional Programming 1 (2): 125–154, April 1991

Introduction to generalized type systems

HENK BARENDREGT
Catholic University Nijmegen, The Netherlands

Abstract

Programming languages often come with type systems. Some of these are simple, others are sophisticated. As a stylistic representation of types in programming languages several versions of typed lambda calculus are studied. During the last 20 years many of these systems have appeared, so there is some need of classification. Working towards a taxonomy, Barendregt (1991) gives a fine-structure of the theory of constructions (Coquand and Huet 1988) in the form of a canonical cube of eight type systems ordered by inclusion. Berardi (1988) and Terlouw (1988) have independently generalized the method of constructing systems in the λ-cube. Moreover, Berardi (1988, 1990) showed that the generalized type systems are flexible enough to describe many logical systems. In that way the well-known *propositions-as-types* interpretation obtains a nice canonical form.

Capsule review

This paper presents a possible classification for the simplest class of typed systems: only β-reduction is considered, and the only type constructors are Π and →. First, various Automath-like typed systems are analysed, with a brief presentation of their main properties (subject reduction, unicity of types and strong normalization).

This analysis suggests rather naturally the notion of *generalized type systems* which provide a neat notation for describing the 'propositions-as-types' idea. The relevance of this notation is shown by the fact that it allows us to sharply express new problems, as the relative completeness of various interpretations, or to state concisely and precisely type-theoretic results (for instance, the exact formalism in which Girard's paradox is derived).

1 Introduction

In several programming languages types are assigned to expressions (occurring in a program) in a way that may be compared to dimensions assigned to entities in physics. These dimensions provide a partial correctness check

$$2\ \text{Volt} + 3\ \text{Ampère}$$

is definitely wrong; the equation

$$E = mc^2$$

is consistent at least from the point of view of dimensions, since both sides are expressed in $\text{kg}.\text{m}^2.\text{sec}^{-2}$.

The analogy between types and dimensions is not perfect. A physical entity always has a unique dimension. Expressions in programming may have more than one type.

This is the case when *implicit* (or *Curry style*) typing is allowed: the expression $\lambda x.x$ denoting the identity function obtains all types $A \to A$ for A an arbitrary type. We write

$$(\lambda x.x):(A \to A)$$

which should be pronounced as 'lambda x dot x *in* A arrow A', and has as its intended meaning that 'for each (element) a in A the application $(\lambda x.x)\,a$ is also in A' (which is intuitively true, since $(\lambda x.x)\,a = a$). Examples of programming languages with this style of typing are ML (Milner 1984) and Miranda (Turner 1985).

There is also another paradigm, the *explicit* or *Church style* of typing, in which each correct expression has exactly one type. Now there are several versions of the identity function

$$\mathbf{I}_A = \lambda x{:}A.x$$

and this one has as its unique type $A \to A$. Examples of languages with explicit typing are LCF (Gordon *et al.* 1979) and TALE (Barendregt and van Leeuwen 1985).

During the last 20 years many systems have appeared for typing lambda calculi, both in the style of Curry and that of Church (see Barendregt (1991) for a survey). In this paper we give some flavour of a class of systems à la Church using the following methodology: *Only the simplest versions of a system are considered*; *that is, only with* β*-reduction, but not with, for example* η*-reduction*; *only with types built up using* $\to$ *and* Π, *not using, for example,* $\times$ *or* Σ. As will be seen, the systems become complicated anyhow. (For a discussion on types in programming languages see Cardelli and Wegner (1985); Reynolds (1985) and Barendregt and Hemerik (1990).)

2 A finestructure of the theory of constructions

Recently a quite powerful typed lambda calculus has been introduced by Coquand and Huet (1988). The system is called 'the theory of constructions', and is denoted here by $\lambda\mathbf{C}$. By analysing the way in which terms and types are built up, a fine-structure of this system is given, consisting of eight systems of typed lambda calculi forming under inclusion a natural cube with oriented edges (see fig. 1). Each edge $\to$ represents the inclusion relation $\subseteq$. This cube is referred to as the λ-*cube*.

Most of the systems in the λ-cube are known, albeit in a somewhat different form. The system $\lambda\to$ is the simply typed lambda calculus (Church, 1940). The system $\lambda\mathbf{2}$ is the *polymorphic* or *second order* typed lambda calculus, and is a subsystem of the system F introduced by Girard (1972). It has been introduced independently by

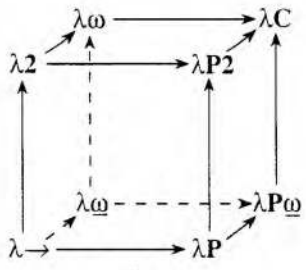


Fig. 1

Reynolds (1974). The system $\lambda\omega$ is essentially the system Fω of Girard (1972). System $\lambda\mathbf{P}$ corresponds reasonably to one of the systems in the family of AUTOMATH languages (see de Bruijn, 1980): System $\lambda\mathbf{P}$ also appears under the name LF in Harper *et al.* (1987). System $\lambda\mathbf{P2}$ is studied by Longo and Moggi (1988) under the same name. System $\lambda\mathbf{C}$ is one of the versions of the theory of constructions introduced by Coquand and Huet (1988), and system $\lambda\underline{\omega}$ is related to the POLYREC system studied by Renardel de Lavalette (1985). System $\lambda\mathbf{P}\underline{\omega}$ seems not to have been studied before. (For $\lambda\underline{\omega}$ and $\lambda\mathbf{P}\underline{\omega}$ read 'weak $\lambda\omega$' and 'weak $\lambda\mathbf{P}\omega$', respectively.)

Before defining the λ-cube, it is useful to describe informally some ideas which play a role in the various systems.

The first informal idea is the construction of function space types. If A and B are types, then $A \to B$ is the type of functions from A to B. So, if $F:(A \to B)$ and $x:A$, then $(Fx):B$.

The second idea is that of dependency. Types and terms are mutually dependent; there are

- terms depending on term;
- terms depending on types;
- types depending on terms;
- types depending on types.

Some explanation is necessary here. Terms depending on terms are extremely common:

$$FM$$

is a term that depends on the term M. A term depending on a type is the identity on A

$$\mathbf{I}_A = \lambda x{:}A.x.$$

A type depending on a term is, for example,

$$A^n \to B$$

(with n a natural number) defined by

$$A^0 \to B = B;$$
$$A^{n+1} \to B = A \to (A^n \to B).$$

And a type depending on a type is, for example,

$$A \to A$$

for A a given type.

Once there are types depending on terms one may introduce cartesian products. Suppose that for each $a:A$ a type B_a is given, and 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 type the cartesian product

$$\Pi a{:}A.B_a$$

of the B_as. Once these product types are allowed, the function space type of A and B can be written as

$$(A \to B) \equiv \Pi a{:}A.B (\equiv B^A, \text{informally}),$$

where a is a variable not occurring in B. This is similar to the fact that a product of equal factors is a power

$$\prod_{i=1}^{n} b_i \text{ becomes } b^n$$

provided that $b_i = b$ for $1 \leqslant i \leqslant n$. So by using products, the type constructor $\to$ can be eliminated.

The next idea has to do with the formation of types. For some simple systems the types are – so to say – given in the metalanguage. For example, if one has informally constructed a type A, then one can formally derive

$$\vdash (\lambda a{:}A.a){:}(A \to A).$$

Since in the λ-cube terms and types are mutually dependent, one moves the formation of types from the metalevel to the formal system itself – the idea comes from the AUTOMATH languages of de Bruijn (1970). To do this a constant $*$ is introduced that is the *sort* of all types; then '$A{:}*$' is a statement expressing 'A is a type'. A sentence in the meta language like

'if A is a type, then so is $A \to A$'

now becomes a formal type derivation

$$A{:}* \vdash (A \to A){:}*.$$

Here A stands for a variable, and since it is in $*$, one can say that A is a type variable.

For each of the four dependencies one may want to introduce function abstraction

$$\begin{aligned}(\lambda m{:}A.Fm){:}(A \to B);\\ (\lambda\alpha{:}*.\mathbf{I}_\alpha){:}(\Pi\alpha{:}*.(\alpha \to \alpha));\\ (\lambda n{:}\mathbb{N}.A^n \to B){:}\mathbb{N} \to *;\\ (\lambda\alpha{:}*.\alpha \to \alpha){:}(* \to *).\end{aligned}$$

Now what is $* \to *$? Probably not a type, because then one should have $(* \to *){:}*$ and this may lead to contradictions. Therefore, one introduces a new 'sort' $\Box$, the sort of all *kinds*, and postulates that $*{:}\Box$ and $(* \to *){:}\Box$. The inhabitants of $* \to *$, like our F, are called *constructors*. Similarly, one postulates $(\mathbb{N} \to *){:}\Box$.

The expression $(\Pi\alpha{:}*.(\alpha \to \alpha))$ being a cartesian product of types will also be a type, so $(\Pi\alpha{:}*.(\alpha \to \alpha)){:}*$. Since it is a product over all possible types α, including the one in statu nascendi (that is, $(\Pi\alpha{:}*.(\alpha \to \alpha))$ is among the types in $*$), there is an essential impredictativity here.

We now start to define the cube of type lambda calculi.

2.1 Definition

(i) The system of the λ-cube are based on a set of pseudo-terms $\mathscr{T}$ defined by the following abstract syntax

$$\mathscr{T} = \mathbf{x} \mid \mathbf{c} \mid \mathscr{T}\mathscr{T} \mid \lambda\mathbf{x}{:}\mathscr{T}.\mathscr{T} \mid \Pi\mathbf{x}{:}\mathscr{T}.\mathscr{T}$$

where $\mathbf{x}$ is the category of variables and $\mathbf{c}$ that of constants.

(ii) On $\mathscr{T}$ the notions of β-conversion and β-reduction and defined by the following contraction rule

$$(\lambda x{:}A.B)C \to_\beta B[x{:=}C]$$

(iii) A *statement* is of the form $A:B$ with $A, B \in \mathscr{T}$. A is the *subject* and B is the *predicate* of $A:B$. A *declaration* is of the form $x:A$ with $A \in \mathscr{T}$ and x a variable. A *pseudo-context* Γ is a finite ordered sequence of declarations, all with distinct subjects. The empty context is denoted by $\langle\,\rangle$. If $\Gamma = \langle x_1:A_1, \ldots, x_n:A_n\rangle$, then

$$\Gamma, x:B = \langle x_1:A_1, \ldots, x_n:A_n, x:B\rangle.$$

Usually we do not write the $\langle\,\rangle$.

(iv) The rules of type assignment will axiomatize the notion

$$\Gamma \vdash A:B$$

stating that $A:B$ can be derived from the context Γ. Pronounce $\Gamma \vdash A:B$ as 'Γ yields A in B'.

The rules are given in two groups: (1) the general rules, valid for all systems of the λ-cube; and (2) the specific rules, differentiating between the eight systems. Two of the constants in C are selected and given the names $*$ and $\Box$. These two constants are called *sorts*. Write $S = \{*, \Box\}$ and let s, s_1, s_2 range over S.

(1) General axiom and rules.

Axiom

$$\langle\,\rangle \vdash *:\Box.$$

Start rule

$$\frac{\Gamma \vdash A:s}{\Gamma, x:A \vdash x:A}, \quad \text{where x is } \Gamma\text{-fresh (x does not occur in } \Gamma).$$

Weakening rule

$$\frac{\Gamma \vdash A:B \quad \Gamma \vdash C:s}{\Gamma, x:C \vdash A:B}, \quad \text{where x is } \Gamma\text{-fresh.}$$

Application rule

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

Conversion rule

$$\frac{\Gamma \vdash A:B \quad \Gamma \vdash B':s \quad B =_\beta B'}{\Gamma \vdash A:B'}.$$

(2) The specific rules are all introduction rules, and are parametrized by two sorts. Let $s_1, s_2 \in S$. Consider the following pair of rules

(s_1, s_2) rules

Π-rule

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

λ-rule

$$\frac{\Gamma \vdash A:s_1 \quad \Gamma, x:A \vdash b:B \quad \Gamma, x:A \vdash B:s_2}{\Gamma \vdash (\lambda x:A.b):(\Pi x:A.B)}$$

(v) *The eight systems of the λ-cube* are defined by taking the general rules plus a specific subset of the set of rule pairs $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$

$\lambda\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
$\lambda\underline{\omega}$	$(*,*)$		$(\Box,\Box)$	
$\lambda\omega$	$(*,*)$	$(\Box,*)$	$(\Box,\Box)$	
$\lambda\mathbf{P}$	$(*,*)$			$(*,\Box)$
$\lambda\mathbf{P2}$	$(*,*)$	$(\Box,*)$		$(*,\Box)$
$\lambda\mathbf{P}\underline{\omega}$	$(*,*)$		$(\Box,\Box)$	$(*,\Box)$
$\lambda\mathbf{P}\omega = \lambda\mathbf{C}$	$(*,*)$	$(\Box,*)$	$(\Box,\Box)$	$(*,\Box)$

The λ-cube will usually be drawn in the following *standard orientation* (see fig. 2). The inclusion relations are left implicit.

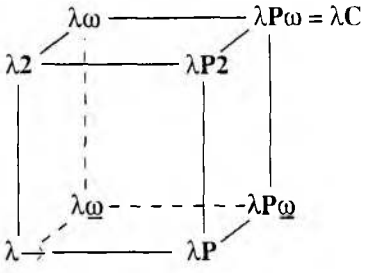


Fig. 2

Notation

(i) Derivability for a system λ_i is denoted by $\Gamma \vdash_{\lambda_i} A:B$. If there is no danger of confusion, or if a statement holds for all systems, then we simply write $\Gamma \vdash A:B$.

(ii) $(A \rightarrow B) \equiv (\Pi x:A.B)$ with $x \notin FV(B)$. This follows the intuition given before.

(iii) $\Gamma \vdash A:B:C$ means $\Gamma \vdash A:B$ and $\Gamma \vdash B:C$.

The rule pairs have the following meaning, as will become clear after studying the examples in section 2.7

$(*,*)$ allows forming terms depending on terms;
$(\Box,*)$ allows forming terms depending on types;
$(*,\Box)$ allows forming terms depending on terms;
$(\Box,\Box)$ allows forming terms depending on types.

2.2 Definition

Consider derivability in one of the systems of the λ-cube.

(i) Let $\Gamma \vdash A:B$. Then A and B are called (legal) *terms* and Γ is called a (legal) *context*.

(ii) Let $\Gamma \vdash A:B:*$. Then A is called an *object* and B a *type*.

(iii) Let $\Gamma \vdash A:B:\Box$. Then A is called a *constructor* and B a *kind*.

It can be shown that a term is an object, a type, a constructor, a kind or a sort. The only overlap is that all types B are also constructors (indeed $B:*:\Box$).

We state some properties about the systems in the λ-cube.

2.3 Church–Rosser Theorem for $\mathcal{T}$
Let A, B, $B' \in \mathcal{T}$. Then

(i) $[A \twoheadrightarrow B \,\&\, A \twoheadrightarrow B'] \Rightarrow \exists C \in \mathcal{T}\,[B \twoheadrightarrow C \,\&\, B' \twoheadrightarrow C]$.
(ii) $A =_\beta B \Rightarrow \exists C \in \mathcal{T}\,[A \twoheadrightarrow C \,\&\, B \twoheadrightarrow C]$.

Proof
Proofs of the Church–Rosser theorem for Λ generalize to $\mathcal{T}$ (see Barendregt and Dekkers, 1990). ■

The following generalizes a result due to Curry *et al.* (1958); see de Vrijer (1975) and van Daalen (1980) for the result in type systems.

2.4 Theorem (subject reduction for the λ*-cube)*
For any system in the λ-cube one has

$$\Gamma \vdash A:B \,\&\, A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash A':B.$$

Proof
See Barendregt (1991) or Geuvers and Nederhof (1991). ■

The following result is due to Coquand. A nice modular proof using the edges of the λ-cube is due to Geuvers and Nederhof (1991).

2.5 Theorem (strong normalization for the λ*-cube)*
For any system in the λ-cube one has

$$\Gamma \vdash A:B \Rightarrow A \text{ and } B \text{ are strongly normalizing,}$$
$$\text{that is all } \beta\text{-reductions starting with A or B terminate.}$$

Proof
See Barendregt (1991) or Geuvers and Nederhof (1991). ■

The following result is folklore.

2.6 Theorem (unicity of types)
For any system in the λ-cube one has

$$\Gamma \vdash A:B \,\&\, \Gamma \vdash A:B' \Rightarrow B =_\beta B'.$$

Proof
See Barendregt (1991) or Geuvers and Nederhof (1991). ■

Some derivable type assignments in the λ*-cube*

We end this subsection by giving for each of the systems in the λ-cube some examples of type assignment. The reader is invited to carefully study these examples in order to gain some intuition in the systems of the λ-cube. Some of the examples are followed by a comment {in brackets}. In order to understand the intended meaning for the systems on the right plane in the λ-cube (that is, the rule pair $(*, \Box)$ is present), some

of the elements of $*$ have to be considered as sets and some as propositions. The examples show that the systems in the λ-cube are related to logical systems and form a preview of the propositions-as-type interpretation described in section 4. Names of variables are chosen freely, in order to follow the intended interpretation.

2.7 Examples

(i) In $\lambda{\to}$ the following can be derived

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

(ii) In $\lambda 2$ the following can be derived

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

of course the following reduction holds

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

The following two examples show a connection with second order proposition logic

$$\vdash (\lambda \beta:* \lambda a:(\Pi \alpha:*.\alpha).a((\Pi \alpha:*.\alpha) \to \beta)\,a):(\Pi \beta:*.(\Pi \alpha:*.\alpha) \to \beta).$$

{For this last example one has to think twice to see that it is correct; a simpler term of the same type in the following; write $\bot \equiv (\Pi \alpha:*.\alpha)$, which is the second order definition of falsum.}

$$\vdash (\lambda \beta:* \lambda a:\bot.a\beta):(\Pi \beta:*.\bot \to \beta).$$

{The type considered as proposition says: 'ex falso sequitur quodlibet'; the term in this type is its proof.}

(iii) In $\lambda\underline{\omega}$ the following can be derived

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

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

$$\begin{aligned}
\beta:* &\vdash (\lambda \alpha:*.\alpha \to \alpha)\,\beta:*;\\
\beta:*, x:\beta &\vdash (\lambda y:\beta.x):(\lambda \alpha:*.\alpha \to \alpha)\,\beta
\end{aligned}$$

{note that $(\lambda y:\beta.x)$ has type $\beta \to \beta$ in the given context}

$$\begin{aligned}
\alpha:*, f:* \to * &\vdash f(f\alpha):*;\\
\alpha:* &\vdash (\lambda f:* \to *.f(f\alpha)):(* \to *) \to *
\end{aligned}$$

{in this way higher order constructors are formed}.

(iv) In $\lambda\mathbf{P}$ the following can be derived

$$A:*\vdash(A\to *):\Box$$

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

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

{if A is a set, $a\in A$ and P is a predicate on A, then Pa is a type considered as proposition (true if inhabited; false otherwise)}

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

{if P is a binary predicate on the set A, then $\forall a\in A$ Paa is a proposition}

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

{this proposition states that the predicate P considered as a set is included in the predicate Q}

$$A:*,P:(A\to *)\vdash(\Pi a:A.(Pa\to Pa)):*$$

{this proposition states the reflexivity of inclusion}

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

{the subject in this assignment provides the 'proof' of reflexivity of inclusion}

$$\begin{aligned}&A:*,P:(A\to *),Q:*\\&\qquad\vdash((\Pi a:A.Pa\to Q)\to(\Pi a:A.Pa)\to Q):*\\&A:*,P:(A\to *),Q:*,a_0:A\\&\qquad\vdash(\lambda x:(\Pi a:A.Pa\to Q)\quad\lambda y:(\Pi a:A.Pa).xa_0(ya_0)):\\&\qquad(\Pi x:(\Pi a:A.Pa\to Q)\to(\Pi y:(\Pi a:A.Pa).\ Q\ )\equiv\\&\qquad\quad(\Pi a:A.Pa\to Q)\to\quad(\Pi a:A.Pa)\to Q\end{aligned}$$

{this proposition states that the proposition $(\forall a\in A.Pa\to Q)\to(\forall a\in A.Pa)\to Q$ is true in non-empty structures. A; notice that the layout explains the functioning of the λ-rule; in this type assignment the subject is the 'proof' of the previous true proposition; note that in the context the assumption $a_0:A$ is needed in this proof.}

(v) In $\lambda\omega$ the following can be derived. Let $\alpha\&\beta\equiv\Pi\gamma:*.(\alpha\to\beta\to\gamma)\to\gamma$, then

$$\alpha:*,\beta:*\vdash\alpha\&\beta:*$$

{this is the 'second order definition of &' and is definable already in $\lambda 2$}. Let $AND\equiv\lambda\alpha:*\lambda\beta:*.\alpha\&\beta$ and $K\equiv\lambda\alpha:*\lambda\beta:*\lambda x:\alpha\lambda y:\beta.x$, then

$$\begin{aligned}&\vdash AND:(*\to *\to *),\\&\vdash K:(\Pi\alpha:*\Pi\beta:*.\alpha\to\beta\to\alpha).\end{aligned}$$

{Note that $\alpha\&\beta$ and K can be derived already in $\lambda 2$, but the term AND cannot}

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

{the subject is a proof that $AND\,\alpha\beta\to\alpha$ is a tautology}.

(vi) In $\lambda\mathbf{P2}$ {corresponding to second order predicate logic} the following can be derived

$$A:*, P:(A \to *) \vdash (\lambda a:A.Pa \to \bot):A \to *$$

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

{the proposition states that a binary relation that is asymmetric is irreflexive}.

(vii) In $\lambda\mathbf{P}\underline{\omega}$ the following can be derived

$$A:* \vdash (\lambda P:(A \to A \to *)\lambda a:A.Paa):((A \to A \to *) \to (A \to *)):\Box$$

{this constructor assigns to a binary predicate P on A its 'diagonalization'}

$$\vdash (\lambda A:*\lambda P:(A \to A \to *)\lambda a:A.Paa):(\Pi A:*\Pi P:(A \to A \to *)\Pi a:A.*):\Box$$

{the same is done uniformly in A}.

(viii) In $\lambda\mathbf{P}\omega = \lambda\mathbf{C}$ the following can be derived

$$\vdash (\lambda A:*\lambda P:(A \to *)\lambda a:A.Pa \to \bot):(\Pi A:*.(A \to *) \to (A \to *)):\Box$$

{this constructor assigns to a type A and to a predicate P on A the negation of P}. Let $ALL \equiv (\lambda A:*\lambda P:A \to *.\Pi a:A.Pa)$; then

$$A:*, P:(A \to *) \vdash ALL\ A\ P:* \text{ and } (ALL\ A\ P) =_\beta (\Pi a:A.Pa)$$

{universal quantification done uniformly}.

Exercises

1. Define $\neg \equiv \lambda\alpha:*.\alpha \to \bot$. Construct a term M such that in $\lambda\omega$

$$\alpha:*, \beta:* \vdash M:((\alpha \to \beta) \to (\neg\beta \to \neg\alpha)).$$

2. Find an expression M such that in $\lambda\mathbf{P2}$

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

3. Find a term M such that in $\lambda\mathbf{C}$

$$A:*, P:(A \to *), a:A \vdash M:(ALL\ A\ P \to Pa).$$

3 Generalized type systems

The method of generating the systems in the λ-cube has been generalized independently by Berardi (1988) and Terlouw (1988). This resulted in the notion of a *generalized type system* (GTS). Many systems of typed lambda calculus à la Church can be seen as GTSs. Subtle differences between systems can be described neatly using the notation of GTSs.

One of the successes of the GTS notion is concerned with logic. In section 4 a cube of eight logical systems is introduced. The systems on this 'logic cube' are in a one-to-one correspondence with the systems on the λ-cube. There is a canonical translation $A \mapsto [A]$ for sentences A such that for a logic $\mathbf{L}_i$ corresponding to a system λ_i on the λ-cube one has

$$\vdash_{\mathbf{L}_j} A \Rightarrow \Gamma \vdash_{\lambda_i} M:[A]$$

for some M canonically depending on the proof of A in $\mathbf{L}_i$; here Γ is some natural context corresponding to the signature of the language in which the logic $\mathbf{L}_i$ is formulated. This result is the so called 'propositions-as-types' interpretation. As was observed by Berardi (1988), the eight logical systems can each be described as a GTSs in such a way that the propositions-as-types interpretation obtains a canonical simple form.

Another reason for introducing GTSs is that several propositions about the systems in the λ-cube are needed. The general setting of the GTSs makes it nicer to give the required proofs.

The generalized type systems are based on the same set of pseudoterms $\mathscr{T}$ for the λ-cube. We repeat the abstract syntax for $\mathscr{T}$

$$\mathscr{T} = \mathbf{x} \mid \mathbf{c} \mid \mathscr{T}\mathscr{T} \mid \lambda \mathbf{x}:\mathscr{T}.\mathscr{T} \mid \Pi \mathbf{x}:\mathscr{T}.\mathscr{T}$$

Let C be the set of constants in $\mathscr{T}$.

3.1 Definition
A *specification of a GTS* consists of a triple $S = (S, A, R)$ where

1. S is a subset of C, called the *sorts*.
2. A is a set of *axioms* of the form $c:s$, with $c \in C$ and $s \in S$.
3. R is a set of rules of the form (s_1, s_2, s_3), with $s_1, s_2, s_3 \in S$.

3.2 Definition
The GTS λS determined by the specification (S, A, R), notation $\lambda S = \lambda(S, A, R)$, is defined as follows. Statements and pseudo-contexts are defined as for the λ-cube. The notion of type derivation $\Gamma \vdash_{\lambda S} A:B$ (we often just write $\Gamma \vdash A:B$) is defined by the following axiom and rules

Axiom
$$\langle\rangle \vdash c:s, \quad \text{if } (c:s) \in A.$$

Start rule
$$\frac{\Gamma \vdash A:s}{\Gamma, x:A \vdash x:A}, \quad \text{where x is fresh.}$$

Weakening rule
$$\frac{\Gamma \vdash B:C \quad \Gamma \vdash A:s}{\Gamma, x:A \vdash B:C}, \quad \text{where x is fresh.}$$

Application rule
$$\frac{\Gamma \vdash F:(\Pi x:A.B):s \quad \Gamma \vdash a:A}{\Gamma \vdash (Fa):(B[x:=a])}$$

Conversion rule
$$\frac{\Gamma \vdash A:B \quad \Gamma \vdash B':s \quad B =_\beta B'}{\Gamma \vdash A:B'}$$

Π-rule
$$\frac{\Gamma \vdash A:s_1 \quad \Gamma, x:A \vdash B:s_2}{\Gamma \vdash (\Pi x:A.B):s_3}, \quad \text{where } (s_1, s_2, s_3) \in R.$$

λ-rule

$$\frac{\Gamma \vdash A:s_1 \quad \Gamma, x:A \vdash B:s_2 \quad \Gamma, x:A \vdash b:B}{\Gamma \vdash (\lambda x:A.b):(\Pi x:A.B)}, \quad \text{where } (s_1, s_2, s_3) \in R \text{ for some } s_3.$$

In the above we use the following conventions

s ranges over S the set of sorts;
x ranges over variables;

'x is fresh' means that x does not occur in Γ, A, B or C.

The proviso $B =_\beta B'$ in the conversion rule is not decidable. However, it can be replaced by the decidable condition

$$B' \to_\beta \beta \text{ or } B \to_\beta B'$$

without changing the set of derivable statements.

3.3 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 simpler form are used.

(ii) The GTS $\lambda(S, A, R)$ is called *full* if

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

(iii) If $\Gamma \vdash A:B:s$, then we say that A is an *element of type* B; if $\Gamma \vdash B:s$, then B is a *type of sort* s.

3.4 Examples

(i) **λP2** is the GTS determined by

$$S = \{*, \Box\}$$
$$A = \{*, \Box\}$$
$$R = \{(*, *), (\Box, *), (*, \Box)\}.$$

Specifications like this will be given more stylistically as follows: $\lambda\mathbf{P2} = \lambda(S, A, R)$ with

λP2

S	$*, \Box$
A	$* : \Box$
R	$(*, *), (\Box, *), (*, \Box)$

(ii) **λC** is the full GTS $\lambda(S, A, R)$ with

λC

S	$*, \Box$
A	$*, \Box$
R	$(*, *), (\Box, *), (*, \Box), (\Box, \Box)$

(iii) A variant $\lambda C'$ of λC is the full GTS $\lambda(S, A, R)$ with

$\lambda C'$

S	$*^{\mathrm{I}}, *^{\mathrm{P}}, \Box$
A	$*^{\mathrm{I}} : \Box, *^{\mathrm{P}} : \Box$
R	S^2, that is all pairs

(iv) $\lambda\!\rightarrow$ is the GTS determined by

$\lambda\!\rightarrow$

S	$*, \Box$
A	$* : \Box$
R	$(*, *)$

(v) A variant of $\lambda\!\rightarrow$, called λ^{τ} in Barendregt (1984—Appendix A) is the GTS determined by

λ^{τ}

S	$*$
A	$\mathbf{0} : *$
R	$(*, *)$

The difference with $\lambda\!\rightarrow$ is that in λ^{τ} no type variables are possible. One only has constant types like $\mathbf{0}, \mathbf{0}\rightarrow\mathbf{0}, \mathbf{0}\rightarrow\mathbf{0}\rightarrow\mathbf{0}, \ldots$ and variables for elements in these types.

(vi) The system $\lambda*$ in which $*$ is the sort of all types, including itself, is specified by

$\lambda*$

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

The system $\lambda*$ is 'inconsistent', in the sense that all types are inhabited. This result is known as Girard's paradox (see for example, Barendregt, 1991). One may think that the inconsistency is caused by the circularity in $*:*$; however Girard (1972) also showed that the following system is inconsistent in the same sense

λU

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

Also, Coquand (1989) showed that λU minus the rule $(\Delta, *)$ is inconsistent.

So far none of the rules was of the form (s_1, s_2, s_3). In section 4 we encounter such rules (in order to represent first order but not higher order functions).

Without proof we mention that the subject reduction theorem holds for all GTSs. The unicity of types theorem does not hold for trivial reasons: there may be two axioms $c:s_1$ and $c:s_2$. The following examples show the flexibility of the notion of GTS.

3.5 Examples (van Benthem Jutting)

Leaving out the definition mechanism, several members of the AUTOMATH family can be exactly described as GTSs. For a description of the systems, see van Daalen, 1980).

(i) The AUT-68 system is described by the following GTS

λAUT-68

S	$*, \Box, \Delta$
A	$*, \Box$
R	$(*, *)$
	$(*, \Box, \Delta), (\Box, *, \Delta), (\Box, \Box, \Delta),$
	$(*, \Delta, \Delta), (\Box, \Delta, \Delta)$

The point is that one may form predicates over a set, but not abstract over them

$A:* \vdash_{\lambda\text{AUT-68}} (A \to *):\Delta;$
$A:* \vdash_{\lambda\text{AUT-68}} (A \to A \to *):\Delta;$
$A:*, a:A, P:(A \to A \to *) \vdash_{\lambda\text{AUT-68}} Paa:*;$
$A:* \vdash_{\lambda\text{AUT-68}} (\lambda F:((A \to A) \to A).F(\lambda x:A.x)):(((A \to A) \to A) \to A).$

Note the correspondence between **λAUT-68** and $\lambda\to$.

(ii) The AUT-QE system is exactly described by the following GTS

λAUT-QE

S	$*, \Box, \Delta$
A	$*, \Box$
R	$(*, *), (*, \Box),$
	$(\Box, *, \Delta), (\Box, \Box, \Delta),$
	$(*, \Delta, \Delta), (\Box, \Delta, \Delta)$

$A:*, a:A \vdash_{\lambda\text{AUT-QE}} (\lambda P:(A \to *).Pa):((A \to *) \to *);$

Note the correspondence between **λAUT-QE** and **λP**.

(iii) The PAL system, a subsystem of AUT-68, is exactly described as follows

λPAL

S	$*, \Box, \Delta$
A	$*, \Box$
R	$(*, *, \Delta), (*, \Box, \Delta), (\Box, *, \Delta),$
	$(\Box, \Box, \Delta),$
	$(*, \Delta, \Delta), (\Box, \Delta, \Delta)$

In this system λ-abstraction is possible only in a restricted way at the 'outside'. However, one may form arbitrary applications

$A:* \vdash_{\lambda PAL} (A \to A):\Delta;$
$A:*, a:A, F:(A \to A) \vdash_{\lambda PAL} Fa:A;$
$A:*, G:(A \to A \to A), a:A, b:A \vdash_{\lambda PAL} Gab:A;$
$A:*, G:(A \to A \to A) \vdash_{\lambda PAL} \lambda b:A \lambda a:A.Gab:(A \to A \to A);$
$A:* \nvdash_{\lambda PAL} \lambda G:(A \to A \to A)\lambda b:A \lambda a:A.Gab:(A \to A \to A) \to (A \to A \to A).$

4 Propositions-as-types

In this section eight systems of intuitionistic logic are introduced—four systems of proposition logic and four systems of many-sorted predicate logic. The systems are the following

PROP proposition logic;
PROP2 second order proposition 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 order predicate logic;
PREDω higher order predicate logic.

All these systems are minimal logics in the sense that the only operators are $\to$ and $\forall$. However, for the second and higher order systems the operations $\neg$, &, $\vee$ and $\exists$, as well as Leibniz's equality, are all definable. Also in these systems one may put in the context $a:(\Pi\alpha:*.\neg\neg\alpha \to \alpha)$ in order to obtain classical logics. Weakly higher order logics have variables for higher order propositions or predicates, but no quantification over them; a higher order proposition has lower order propositions as arguments.

The systems form a cube as shown in fig. 3. This cube is referred to as the L-cube. The orientation of the L-cube as drawn is called the standard orientation. Each system L_i on the L-cube corresponds to the system λ_i on the λ-cube on the corresponding vertex (both cubes in standard orientation). The edges of the L-cube represent inclusions of systems in the same way as on the λ-cube.

A formula A in the logic L_i on the L-cube can be interpreted as a type $[\![A]\!]$ in the

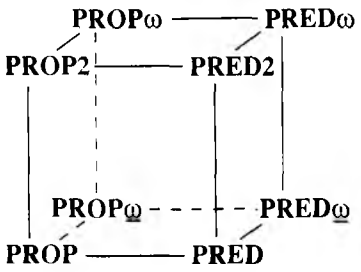


Fig. 3

corresponding λ_i on the λ-cube. The transition $A \mapsto [\![A]\!]$ is called the propositions-as-types interpretation of de Bruijn (1970) and Howard (1980), first formulated for extensions of **PRED** and $\lambda\mathbf{P}$. The method has been extended by Martin-Löf (1984), who added to $\lambda\mathbf{P}$ types $\Sigma x{:}A.B$ corresponding to (strong) constructive existence and a constructor $=_A : A \to A \to *$ corresponding to equality on a type A. Since Martin-Löf's principle objective is to give a constructive foundation of mathematics, he does not consider the impredicative rules $(\Box, *)$.

This interpretation satisfies the following soundness result: if A is provable in **PRED**, then $[\![A]\!]$ is inhabited in $\lambda\mathbf{P}$. In fact, an inhabitant of $[\![A]\!]$ in $\lambda\mathbf{P}$ can be found canonically from a proof of A in **PRED**; different proofs of A are interpreted as different terms of type $[\![A]\!]$.

The propositions-as-types interpretation has been extended to several other systems (for example, see Martin-Löf 1984 and Stenlund 1972). In Geuvers (1988) it is verified that for all systems L_i on the L-cube soundness holds with respect to the corresponding system λ_i on the λ-cube: if A is probable in L_i then $[\![A]\!]$ is inhabited in λ_i. Barendsen (1989) verifies that a proof D of such A can be canonically translated to $[\![D]\!]$ being an inhabitant of $[\![A]\!]$.

After seeing Geuvers (1988), it was realised by Berardi (1988; 1990) that the systems in the L-cube can be considered as GTSs. Doing this the propositions-as-types interpretation obtains a simple canonical form. We first give a description of **PRED** in its usual form, and then in its form as a GTS.

The soundness result for the propositions-as-type interpretation raises the question whether one also has completeness in the sense that if given a formula A of a logic L_i is such that $[\![A]\!]$ is inhabited in λ_i then A is probable in L_i.

For the proposition logics this is trivially true, for **PRED** completeness with respect to $\lambda\mathbf{P}$ is proved by Martin-Löf (1970), Barendsen and Geuvers (1989) and Berardi (1990) (see also Swaen 1989). For **PRED**ω completeness with respect to λC fails, as is shown by Geuvers (1989) and Berardi (1990).

Many sorted predicate logic

4.1 Definition

The notion of a *many sorted structure* is defined by an example. The following sequence is a typical many sorted structure

$$\mathcal{A} = \langle A, B, f, g, P, Q, c \rangle$$

with A, B are non-empty sets, the *sorts* of $\mathscr{A}$ (we use the standard terminology; in the context of GTSs it would be better to call A and B 'types'); $f:(A \to (A \to A))$ and $g:A \to B$ are functions; $P \subseteq A$ and $Q \subseteq A \times B$ are relations; and $c \in A$ is a constant.

4.2 Definition
Given the many sorted structure $\mathscr{A}$ of Section 4.1, the language $L_{\mathscr{A}}$ of minimal many sorted predicate logic over $\mathscr{A}$ is defined as follows

(i) $L_{\mathscr{A}}$ has the following special symbols

$\mathbb{A}, \mathbb{B}$	sort symbols;
$\mathbb{P}, \mathbb{Q}$	relation symbols;
$\mathbb{f}, \mathbb{g}$	function symbols;
$\mathbb{c}$	constant symbol.

(ii) The set of variables of $L_{\mathscr{A}}$ is

$$V = \{x^{\mathbb{A}} \mid x \text{ variable}\} \cup \{x^{\mathbb{B}} \mid x \text{ variable}\}.$$

(iii) The set of terms of sort A and of sort B, notation Term_A and Term_B respectively, are defined inductively as follows

$$x^{\mathbb{A}} \in \mathrm{Term}_A,\ x^{\mathbb{B}} \in \mathrm{Term}_B;$$
$$\mathbb{c} \in \mathrm{Term}_A;$$
$$s \in \mathrm{Term}_A \text{ and } t \in \mathrm{Term}_A \Rightarrow \mathbb{f}(s, t) \in \mathrm{Term}_A;$$
$$s \in \mathrm{Term}_A \Rightarrow \mathbb{g}(s) \in \mathrm{Term}_B.$$

(iv) The set of formulas of $L_{\mathscr{A}}$, notation Form, is defined inductively as follows

$$s \in \mathrm{Term}_A,\ t \in \mathrm{Term}_B \Rightarrow \mathbb{P}(s), \mathbb{Q}(s, t) \in \mathrm{Form};$$
$$\varphi \in \mathrm{Form},\ \psi \in \mathrm{Form} \Rightarrow (\varphi \to \psi) \in \mathrm{Form};$$
$$\varphi \in \mathrm{Form} \Rightarrow (\forall x^{\mathbb{A}}.\varphi) \in \mathrm{Form} \text{ and } (\forall x^{\mathbb{B}}.\varphi) \in \mathrm{Form}.$$

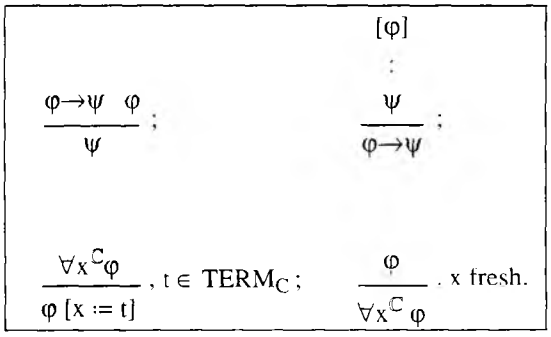
4.3 Definition
Let $\mathscr{A}$ be a many sorted structure. The minimal many sorted predicate logic over $\mathscr{A}$, notation $\mathbf{PRED} = \mathbf{PRED}_{\mathscr{A}}$, is defined as follows. If Δ is a set of formulas, then $\Delta \vdash \varphi$ denotes that φ is derivable from assumptions Δ. This notion is defined inductively as follows (C ranges over A and **B**, and the corresponding $\mathbb{C}$ over $\mathbb{A}, \mathbb{B}$)

$$\varphi \in \Gamma \Rightarrow \Gamma \vdash \varphi$$
$$\Gamma \vdash \varphi \to \psi,\ \Gamma \vdash \varphi \Rightarrow \Gamma \vdash \psi$$
$$\Gamma, \varphi \vdash \psi \Rightarrow \Gamma \vdash \varphi \to \psi$$
$$\Gamma \vdash \forall x^{C}.\varphi,\ t \in \mathrm{Term}_C \Rightarrow \Gamma \vdash \varphi[x := t]$$
$$\Gamma \vdash \varphi,\ x^{C} \notin \mathrm{FV}(\Gamma) \Rightarrow \Gamma \vdash \forall x^{C}.\varphi,$$

where $[x := t]$ denotes substitution of t for x, and FV is the set of free variables in a term, formula or collection of formulas.

For $\varnothing \vdash \varphi$ one writes simply $\vdash \varphi$ and one says that φ is a *theorem*.

These rules can be remembered best in the form of the following natural deduction form



Some examples of terms, formulas and theorems are the following. The expressions x^A, c, $f(x^A, c)$ and $f(c, c)$ are all in Term_A; $g(x^A)$ is in Term_B. Moreover

$$\forall x^A \mathbb{P}(f(x^A, x^A)), \tag{1}$$

$$\forall x^A [\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c)], \tag{2}$$

$$\forall x^A [\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c)] \to \forall x^A \mathbb{P}(x^A) \to \mathbb{P}(f(c, c) \tag{3}$$

are formulas. The formula (3) is even a theorem. A derivation of (3) is as follows

$$\dfrac{\dfrac{\dfrac{[\forall x^A[\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c))]]\,2 \quad [\forall x^A \mathbb{P}(x^A)]\,1}{\mathbb{P}(c) \to \mathbb{P}(f(c, c))} \quad \dfrac{}{\mathbb{P}(c)}}{\dfrac{\mathbb{P}(f(c, c))}{\forall x^A \mathbb{P}(x^A) \to \mathbb{P}(f(c, c))}\,1}}{\forall x^A[\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c))] \to \forall x^A \mathbb{P}(x^A) \to \mathbb{P}(f(c, c))}\,2$$

the numbers 1, 2 indicating when a cancellation of an assumption is being made. A simpler derivation of the same formula is

$$\dfrac{[\forall x^A(\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c)]\,2 \quad \dfrac{\dfrac{[\forall x^A \mathbb{P}(x^A)]\,1}{\mathbb{P}(f(c, c)}}{\forall x^A \mathbb{P}(x^A) \to \mathbb{P}(f(c, c))}\,1}{\forall x^A(\mathbb{P}(x^A) \to \mathbb{P}(f(x^A, c)) \to \forall x^A(\mathbb{P}(x^A) \to \mathbb{P}(f(c, c))}\,2$$

Now we explain, first somewhat informally, the propositions-as-types interpretation form **PRED** into $\lambda\mathbf{P}$. First one needs a context corresponding to the structure $\mathscr{A}$. This is $\Gamma_{\mathscr{A}}$ defined as follows (later $\Gamma_{\mathscr{A}}$ is defined as little differently)

$$\begin{aligned}\Gamma_{\mathscr{A}} = {} & A:*, B:*, \\ & P:A \to *, Q:A \to B \to *, \\ & f:A \to A \to A, g:A \to B, \\ & c:A.\end{aligned}$$

For this context one has

$$\Gamma_{\mathscr{A}} \vdash c : A \tag{0'}$$
$$\Gamma_{\mathscr{A}} \vdash (fcc) : A$$
$$\Gamma_{\mathscr{A}} \vdash [\Pi x : A . P(fxx)] : * \tag{1'}$$
$$\Gamma_{\mathscr{A}} \vdash [\Pi x : A . (Px \to P(fxc))] : * \tag{2'}$$
$$\Gamma_{\mathscr{A}} \vdash [[\Pi x : A . (Px \to P(fxc))] \to [(\Pi x : A . Px) \to P(fcc)]] : *. \tag{3'}$$

We see how the formulas (1) to (3) are translated as types. The inhabitants of $*$ have a somewhat 'ambivalent' behaviour, they serve both as sets (for example, $A : *$), and as propositions (for example, $Px : *$ for $x : A$). The fact that formulas are translated as types is called the *propositions-as-types* (or also *formulas-as-types*) interpretation. The provability of formula (3) corresponds to the fact that the type in (3′) is inhabited. In fact

$$\Gamma_{\mathscr{A}} \vdash \lambda p : [\Pi x : A . (Px \to P(fxc))] \lambda q : (\Pi x : A . Px) . pc(qc) :$$
$$\Pi p : [\Pi x : A . (Px \to P(fxc))] \Pi q : (\Pi x : A . Px) . P(fcc).$$

A somewhat simpler inhabitant of the type in (3′), corresponding to the second proof of the formula (3), is

$$\lambda P : [\Pi x : A . (Px \to P(fxc))] \lambda q : (\Pi x : A . Px) . q(fcc).$$

In fact, one has the following result, which at the moment we state informally (and which in fact, is not completely correct; therefore, no number is given to the item).

Theorem (*soundness of the propositions-as-types interpretation*)
Let $\mathscr{A}$ be a many sorted structure and let φ be a formula of $L_{\mathscr{A}}$. Suppose

$$\vdash_{\mathbf{PRED}} \varphi \text{ with derivation } D;$$

then

$$\Gamma_{\mathscr{A}} \vdash_{\lambda \mathbf{P}} [D] : [\varphi] : *,$$

where $[D]$ and $[\varphi]$ are canonical translations of φ and D, respectively.

Now we show that **PRED** can be viewed as a GTS, and then it follows that the map $\varphi \mapsto [\varphi]$ can be factorized as a composition of an isomorphism **PRED** $\to \lambda$**PRED** and a canonical forgetful homomorphism λ**PRED** $\to \lambda \mathbf{P}$/

4.4 Definition (*Berardi* 1988)
PRED considered as a GTS, notation λ**PRED**, is determined by the following specification

λPRED

S	$*^s, *^p, *^f, \Box^s, \Box^p$
A	$*^s : \Box^s, *^p : \Box^p$
R	$(*^p, *^p), (*^s, *^p), (*^s, \Box^p),$ $(*^s, *^s, *^f), (*^s, *^f, *^f)$

Some explanations are necessary here. The sort $*^s$ is for sets (the 'sorts' of the many sorted logic). The sort $*^p$ is for propositions (the formulas of the logic will become elements of $*^p$). The sort $*^f$ is for first order functions between the sets in $*^s$. The sort $\Box^s$ contains $*^s$, and the sort $\Box^p$ contains $*^p$. (There is no $\Box^f$, otherwise it would be allowed to have free variables for function spaces.) The rule $(*^p, *^p)$ allows the formation of implication of two formulas

$$\varphi : *^p, \psi : *^p \vdash (\varphi \to \psi) \equiv (\Pi x : \varphi . \psi) : *^p.$$

The rule $(*^s, *^p)$ allows quantification over sets:

$$A : *^s, \varphi : *^p \vdash `(\forall x^{A} . \varphi)' \equiv (\Pi x : A . \varphi) : *^p.$$

The rule $(*^s, \Box^p)$ allows the formation of first order predicates:

$$A : *^s \vdash (A \to *^p) \equiv (\Pi x : A . *^p) : \Box^p;$$

hence

$$A : *^s, x : A, P : (A \to *^p) \vdash Px : *^p,$$

that is, P is a predicate over the set A.

The rule $(*^s, *^s, *^f)$ allows the formation of a function space between the basic sets in $*^s$

$$A : *^s, B : *^s \vdash (A \to B) : *^f;$$

the rule $(*^s, *^f, *^f)$ allows the formation of curried functions of several arguments in the basic sets

$$A : *^s \vdash (A \to (A \to A)) : *^f$$

This makes is possible to have, for example, $g : A \to B$ and $f : (A \to (A \to A))$ in a context.

Now it is shown that λ**PRED** is able to simulate the logic **PRED**. Terms, formulas and derivations of **PRED** are translated into terms of λ**PRED**. Terms become elements, formulas become types and a derivation of a formula φ becomes an element of the type corresponding to φ.

4.5 Definition

Let $\mathscr{A}$ be as in Section 4.1. the *canonical context* corresponding to $\mathscr{A}$, notation $\Gamma_{\mathscr{A}}$, is defined by

$$\begin{aligned} \Gamma_{\mathscr{A}} = {} & A : *^s, B : *^s, \\ & P : B \to *^p, Q : A \to B \to *^p, \\ & f : A \to (A \to B), g : A \to B, \\ & c : B. \end{aligned}$$

Given a term $t \in \Gamma_{\mathscr{A}}$, the canonical translation of t, notation $[\![t]\!]$, and the canonical context for t, notation Γ_t, are inductively defined as follows

t	$[\![t]\!]$	Γ_t
x^C	x	x : C
c	c	$\langle\ \rangle$
f (s, s')	f $[\![s]\!]$ $[\![s']\!]$	$\Gamma_s \cup \Gamma_{s'}$
g (s)	g $[\![s]\!]$	Γ_s

Given a formula φ in $L_{\mathscr{A}}$, the canonical translation of φ, notation $[\![\varphi]\!]$, and the canonical context for φ, notation Γ_φ are inductively defined as follows

φ	$[\![\varphi]\!]$	Γ_φ
$\mathcal{P}(t)$	$P\,[\![t]\!]$	Γ_t
$\mathcal{Q}(s, t)$	$Q\,[\![s]\!]\,[\![t]\!]$	$\Gamma_s \cup \Gamma_t$
$\varphi_1 \to \varphi_2$	$[\![\varphi_1]\!] \to [\![\varphi_2]\!]$	$\Gamma_{\varphi_1} \cup \Gamma_{\varphi_2}$
$\forall x^{\mathbb{C}}.\,\psi$	$\Pi x{:}C\,.\,[\![\psi]\!]$	$\Gamma_\psi - \{x{:}C\}$

4.6 Lemma

(i) $t \in \mathrm{TERM}_{\mathbb{A}} \Rightarrow \Gamma_{\mathscr{A}}, \Gamma_t \vdash_{\lambda\mathbf{PRED}} [\![t]\!] : A$; similarly for B.

(ii) $\varphi \in \mathrm{FORM} \Rightarrow \Gamma_{\mathscr{A}}, \Gamma_\varphi \vdash_{\lambda\mathbf{PRED}} [\![\varphi]\!] : *^{\mathrm{P}}$.

Proof

By an easy induction. ■

In order to define the canonical translation of derivations, it is useful to introduce some notation. The following definition is a reformulation of definition 4.3, now giving formal notations for derivations.

4.7 Definition

In **PRED** the notation 'D is a derivation showing $\Delta \vdash \varphi$', notation $D{:}\Delta \vdash \varphi$, is defined as follows

$$\varphi \in \Delta \Rightarrow \mathbf{P}_\varphi : \Delta \vdash \varphi;$$
$$D_1 : \Delta \vdash \varphi \to \psi,\ D_2 : \Delta \vdash \varphi \Rightarrow (D_1\,D_2) : \Delta \vdash \psi;$$
$$D : \Delta, \varphi \vdash \psi \Rightarrow (\mathrm{I}\varphi).\,D : \Delta \vdash \varphi \to \psi;$$
$$D : \Delta \vdash \forall x^{\mathbb{C}}.\varphi,\ t \in \mathrm{TERM}_{\mathbb{C}} \Rightarrow (Dt) : \Delta \vdash \varphi[x := t];$$
$$D : \Delta \vdash \varphi,\ x^{\mathbb{C}} \notin \mathrm{FV}(\Delta) \Rightarrow (\mathrm{G}x^{\mathbb{C}}\Delta) : D \vdash \forall x^{\mathbb{C}}.\varphi.$$

Here $\mathbb{C}$ is $\mathbb{A}$ or $\mathbb{B}$, **P** stands for 'projection', $\mathrm{I}\varphi$ stands for introduction and has a binding effect on φ, and $\mathrm{G}x^{\mathbb{C}}$ stands for 'generalization' (over C) and has a binding effect on $x^{\mathbb{C}}$.

4.8 Definition

(i) Let $\Delta = \{\varphi_1, \ldots, \varphi_n\} \subseteq \mathrm{FORM}$. Then the canonical translation of Δ, notation Γ_Δ, is the context defined by

$$\Gamma_\Delta = \Gamma_{\varphi_1} \cup \ldots \cup \Gamma_{\varphi_n}, x_{\varphi_1} : [\![\varphi_1]\!], \ldots, x_{\varphi_n} : [\![\varphi_n]\!].$$

(ii) For $D{:}\Delta \vdash \varphi$ in **PRED** the canonical translation of D, notation $[\![D]\!]$, and the canonical context for D, notation Γ_D, are inductively defined as follows

D	$[\![D]\!]$	Γ_D
P_φ	x_φ	$\langle\ \rangle$
$D_1\,D_2$	$[\![D_1]\!]\,[\![D_2]\!]$	$\Gamma_{D_1} \cup \Gamma_{D_2}$
$\mathrm{I}\varphi\,.\,D_1$	$\lambda x_\varphi{:}[\![\varphi]\!].[\![D_1]\!]$	$\Gamma_{D_1} - \{x_\varphi : [\![\varphi]\!]\}$
Dt	$[\![D]\!]\,[\![t]\!]$	$\Gamma_D \cup \Gamma_t$
$\mathrm{G}x^{\mathbb{C}}.\,D$	$\lambda x{:}C.[\![D]\!]$	$\Gamma_D - \{x{:}C\}$

4.9 Lemma

$$D:\Delta \vdash_{\mathbf{PRED}} \varphi \Rightarrow \Gamma_{\mathscr{A}}, \Gamma_{\Delta} \cup \Gamma_{\varphi} \cup \varphi_D \vdash_{\lambda\mathbf{PRED}} [\![D]\!]:[\![\varphi]\!].$$

Proof
By induction on the derivation in **PRED**. ■

The following lemma is a kind of converse lemma 4.9.

4.10 Lemma (*K. Fujita* 1989)
Suppose $\Gamma \vdash_{\lambda\mathbf{PRED}} A:B:*^p$. Then there is a many sorted structure $\mathscr{A}$, a set of formulas $\Delta \subseteq L_{\mathscr{A}}$, a formula $\varphi \in L_{\mathscr{A}}$ and a derivation D such that

$$\Gamma \equiv \Gamma_{\mathscr{A}}, \Gamma_{\Delta} \cup \Gamma_{\varphi} \cup \Gamma_D,$$
$$A \equiv [\![D]\!], B \equiv [\![\varphi]\!]$$
$$D:\Delta \vdash_{\mathbf{PRED}} \varphi. \quad \blacksquare$$

The following result gives the soundness of the interpretation $[\![\ ]\!]$. Note, however, that, for example, a sentence φ, that is, $FV(\varphi) = \varnothing$, one has in general

$$\vdash_{\mathbf{PRED}} \varphi \not\Rightarrow \exists A \Gamma_{\mathscr{A}} \vdash_{\lambda\mathbf{PRED}} A:[\![\varphi]\!].$$

The reason is that logic is such that it assumes that the intended domains are non-empty. For example

$$(\forall x^A.(Px \to Q)) \to ((\forall x^A.Px) \to Q)$$

is provable in **PRED**, but only valid in structures with $A \neq \varnothing$.

4.11 Definition
The *extended context* $\Gamma^+_{\mathscr{A}}$ is defined by $\Gamma^+_{\mathscr{A}} = \Gamma_{\mathscr{A}}, a:A, b:B$.

So, $\Gamma^+_{\mathscr{A}}$ explicitly states that the domains in question are not empty. Now one does have completeness.

4.12 Corollary
(i) Let φ be a formula and Δ be a set of formulas of $L_{\mathscr{A}}$. Then

$$D:\Delta \vdash_{\mathbf{PRED}} \varphi \Leftrightarrow \Gamma_{\mathscr{A}}, \Gamma_{\Delta} \cup \Gamma_{\varphi} \cup \Gamma_D \vdash_{\lambda\mathbf{PRED}} [\![D]\!]:[\![\varphi]\!].$$

(ii) Let $\Delta \cup \{\varphi\}$ be a set of sentences of L_{σ}. Then

$$\Delta \vdash_{\mathbf{PRED}} \varphi \Leftrightarrow \Gamma^+_{\mathscr{A}}, \Gamma_{\Delta} \vdash_{\lambda\mathbf{PRED}} M:[\![\varphi]\!] \text{ for some } M.$$

(iii) Let φ be a sentence of $L_{\mathscr{A}}$. Then

$$\vdash_{\mathbf{PRED}} \varphi \Leftrightarrow \exists M\, \Gamma^+_{\mathscr{A}} \vdash_{\lambda\mathbf{PRED}} M:[\![\varphi]\!].$$

Proof
(i) By definition 4.9 and 4.10, and the fact that $[\![\ ]\!]$ is injective on derivation and formulas.
(ii) If the members of Δ and φ are without free variables, then

$$D:\Delta \vdash_{\mathbf{PRED}} \varphi \Leftrightarrow \Gamma_{\mathscr{A}}, \Gamma_{\Delta} \cup \Gamma_D \vdash_{\lambda\mathbf{PRED}} [\![D]\!]:[\![\varphi]\!]$$

A statement in Γ_D is of the form $x:C$. Since $\Gamma^+_{\mathscr{A}} \vdash a:A, b:B$ one has

$$\begin{aligned}\Delta \vdash_{\mathbf{PRED}} \varphi &\Leftrightarrow \exists D\, D:\Delta \vdash_{\mathbf{PRED}} \varphi \\ &\Leftrightarrow \exists D\, \Gamma_{\mathscr{A}}, \Gamma_\Delta \cup \Gamma_D \vdash_{\lambda\mathbf{PRED}} [\![D]\!]:[\![\varphi]\!] \\ &\Leftrightarrow \exists M\, \Gamma^+_{\mathscr{A}}, \Gamma_\Delta \vdash \lambda_{\mathbf{PRED}} M:[\![\varphi]\!].\end{aligned}$$

(For the last $\Rightarrow$ take $M \equiv [\![D]\!][x, y := a, b]$; for $\Leftarrow$ use lemma 4.10).

(iii) By (ii), taking $\Delta = \varnothing$. ■

The system λ**PRED** is also flexible enough to cover so-called *free logic* with empty domains as developed by Peremans (1949) and Mostowski (1951); simply work in context $\Gamma_{\mathscr{A}}$ instead of $\Gamma^+_{\mathscr{A}}$.

Now that it has been established that **PRED** and λ**PRED** are isomorphic, the propositions-as-types interpretation from **PRED** to λ**P** can be factorized in two simple steps: from **PRED** to λ**PRED** via the isomorphism and from λ**PRED** to λ**P** via a canonical forgetful map.

4.13 Definition (*propositions-as-types interpretation*)
(i) Define the forgetful map $|\ |$: term (λ**PRED**) $\to$ term (λ**P**) be deleting all superscripts in $*$ and $\Box$

$$\begin{aligned}*^s &\mapsto * \\ *^p &\mapsto * \\ \Box^s &\mapsto \Box \\ \Box^p &\mapsto \Box.\end{aligned}$$

for example, $|\lambda x:*^p.x| \equiv \lambda x:*.x$. Write $|\Gamma| \equiv \{x_1:|A_1|, \ldots\}$ for $\Gamma \equiv \{x_1:A_1, \ldots\}$.
(ii) Let $\mathscr{A}$ be a signature and let t, φ, Δ and D be, respectively, a term, a formula, a set of formulas and a derivation in **PRED** formulated in $L_{\mathscr{A}}$. Write

$$\begin{aligned}[t] &= |[\![t]\!]|; \\ [\varphi] &= |[\![\varphi]\!]|; \\ [D] &= |[\![D]\!]|; \\ [\Delta] &= |\Gamma^+_{\mathscr{A}}|, |\Gamma_\Delta|.\end{aligned}$$

4.14 Corollary (*Soundedness for the propositions-as-types interpretation*)
(i) $\Gamma \vdash_{\lambda\mathbf{PRED}} A:B \Rightarrow |\Gamma| \vdash_{\lambda\mathbf{P}} |A|:|B|$.
(ii) For sentences Δ and φ in L_A one has

$$D:\Delta \vdash_{\mathbf{PRED}} \varphi \Rightarrow [\Delta] \vdash_{\lambda\mathbf{P}} M:[\varphi], \text{ for some } M.$$

Proof
(i) By a trivial induction on derivations in λ**PRED**.
(ii) By corollary 4.12(ii) and (i). ■

As was remarked before, the converse, completeness for the propositions-as-types interpretation holds for **PRED** and λ**P**, but not for **PRED**ω and λ**C**.

4.15 Theorem (*Berardi* 1989; *Geuvers*, 1989)
Consider the similarity type of the structure $\mathscr{A} = \langle A \rangle$, i.e. there is one set without any relations. Then there is in the signature of $\mathscr{A}$ a sentence φ of **PRED**ω such that

$$\nvdash_{\mathbf{PRED}\omega} \varphi$$

but for some M one has

$$\Gamma_{\mathscr{A}} \vdash_{\lambda C} M : [\varphi].$$

Proof sketch (*Bernardi*)
Define

$$\mathrm{EXT} \equiv \Pi p : * \Pi p' : * . [(p \leftrightarrow p') \to \Pi Q : * \to * . (Qp \to Qp')]$$
$$\varphi \equiv \mathrm{EXT} \to \text{'A does not have exactly two elements'}$$

Obviously, $\nvdash_{\mathbf{PRED}\omega} \varphi$. Claim: interpreted in $\lambda\mathbf{C}$ one has

$$\mathrm{EXT} \to \text{'if A is non-empty, then A is a type-free } \lambda\text{-model'}.$$

The reason is that if $a : A$, then

$$\vdash (\lambda x : (A \to A) . a) : ((A \to A) \to A)$$

and always

$$\vdash (\lambda y : A \lambda z : A . z) : (A \to (A \to A)),$$

therefore, '$A \leftrightarrow (A \to A)$' and since '$A \cong A$' (that is, there is a bijection from A to A), it follows by EXT that '$A \cong (A \to A)$', that is, 'A is a type-free λ-model'.

By the claim A cannot have two elements, since only the trivial λ-model is finite. ■

The counterexample of Geuvers is technically simpler, but intuitively somewhat more complicated; it is also related to the statement EXT.

The definition of the other systems in the λ-cube is now given. After having seen the equivalent between **PRED** and λ**PRED**, each system is described directly as a GTS and not as a more traditional logical system.

4.16 Definition
(i) Systems λ**PROP**, λ**PROP2**, λ**PROP**$\underline{\omega}$ and λ**PROP**ω are the GTSs specified as follows

λPROP

S	$*^p, \Box^p$
A	$*^p : \Box^p$
R	$(*^p, *^p)$

λPROP2 = **λPROP** + $(\Box^p, *^p)$

S	$*^p, \Box^p$
A	$*^p : \Box^p$
R	$(*^p, *^p), (\Box^p, *^p)$

λPROPω = λPROP + $(\Box^P, \Box^P)$

S	$*^P$, $\Box^P$
A	$*^P$, $\Box^P$
R	$(*^P, *^P)$, $(\Box^P, \Box^P)$

λPROPω = λPROP + $(\Box^P, *^P)$ + $(\Box^P, \Box^P)$

S	$*^P$, $\Box^P$
A	$*^P$, $\Box^P$
R	$(*^P, *^P)$, $(\Box^P, *^P)$, $(\Box^P, \Box^P)$

(ii) Systems **λPRED**, **λPRED2**, **λPREDω** and **λPREDω** are the GTSs specified as follows

λPRED

S	$*^P$, $*^S$, $*^f$, $\Box^P$, $\Box^S$
A	$*^P$, $\Box^P$, $*^S$, $\Box^S$
R	$(*^P, *^P)$, $(*^S, *^P)$, $(*^S, \Box^P)$
	$(*^S, *^S, *^f)$, $(*^S, *^f, *^f)$

λPRED2 = λPRED + $(\Box^P, *^P)$

S	$*^P$, $*^S$, $*^f$, $\Box^P$, $\Box^S$
A	$*^P$, $\Box^P$, $*^S$, $\Box^S$
R	$(*^P, *^P)$, $(*^S, *^P)$, $(*^S, \Box^P)$
	$(*^S, *^S, *^f)$, $(*^S, *^f, *^f)$
	$(\Box^P, *^P)$

λPREDω = λPRED + $(\Box^P, \Box^P)$

S	$*^P$, $*^S$, $*^f$, $\Box^P$, $\Box^S$
A	$*^P$, $\Box^P$, $*^S$, $\Box^S$
R	$(*^P, *^P)$ $(*^S, *^P)$, $(*^S, \Box^P)$
	$(*^S, *^S, *^f)$, $(*^S, *^f, *^f)$
	$(\Box^P, \Box^P)$

$\lambda\mathbf{PRED}\omega = \lambda\mathbf{PRED} + (\Box^P, *^P) + (\Box^P, \Box^P)$

S	$*^P, *^S, *^f, \Box^P, \Box^S$
A	$*^P, \Box^P, *^S, \Box^S$
R	$(*^P, *^P)\ (*^S, *^P), (*^S, \Box^P)$
	$(*^S, *^S, *^f), (*^S, *^f, *^f)$
	$(\Box^P, *^P), (\Box^P, \Box^P)$

The eight systems form a cube as shown in fig. 4.

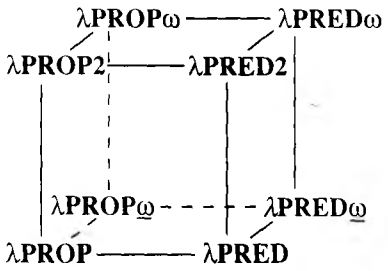


Fig. 4

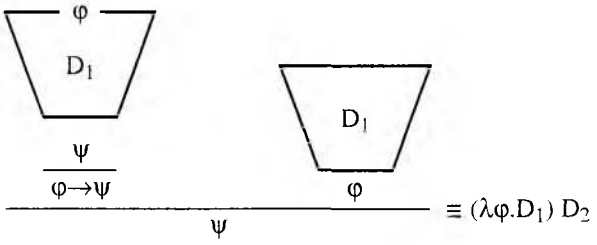
Since the description of these GTSs is more uniform than the original description of the logical systems, this cube will be considered as the L-cube. In particular, fig. 4 displays the standard orientation of the L-cube, and each L_i (ranging over $\lambda\mathbf{PROP}$, $\lambda\mathbf{PRED}$, etc.) corresponds to a unique system λ_i on the similar vertex in the λ-cube (in standard orientation).

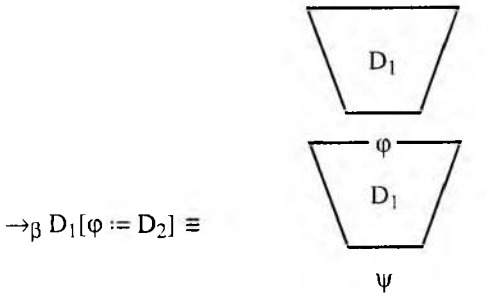
4.17 Theorem (*soundness of the propositions-as-types interpretation*)
Let L_i be a system on the L-cube, and let λ_i be the corresponding system on the λ-cube. The forgetful map $|\ |$ that erases all superscripts in the $*^s$ and $\Box^s$ satisfies the following

$$\Gamma \vdash_{L_i} A:B:s \Rightarrow |\Gamma| \vdash_{\lambda_i} |A|:|B|:|s|. \tag{4}$$

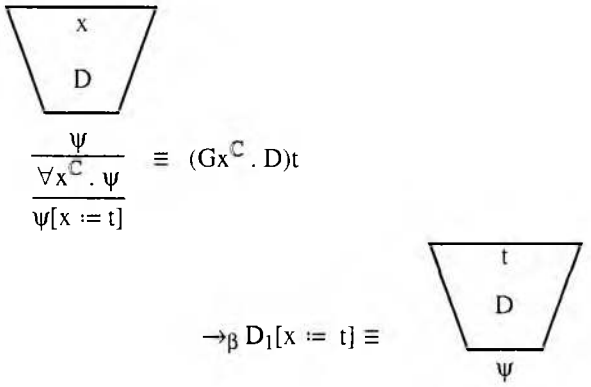
Proof
By a trivial induction on the derivation in L_i. ■

As is well-known, logical deductions are subject to reduction (for example, see Prawitz 1965; or Stenlund 1972). For example, in **PRED** one has





and



If the deductions are represented in λ**PRED**, then these reductions become ordinary β-reductions

$$[\![(\lambda\varphi\,.\,D_1)\,D_2]\!] \equiv (\lambda x{:}[\![\varphi]\!]\,.\,[\![D_1]\!])\,[\![D_2]\!] \to_\beta [\![D_1]\!][x := [\![D_2]\!]] \equiv [\![D_1[x := D_2]]\!];$$

$$[\![(Gx^C\,.\,D)\,t]\!] \equiv (\lambda x{:}C\,.\,[\![D]\!])\,[\![t]\!] \to_\beta [\![D]\!][x := [\![t]\!]] \equiv [\![D[x := t]]\!]$$

In fact, the best way to define the notion of reduction for a logical system on the L-cube is to consider that system as a GTS subject to β-reductions.

Now it follows that reductions in all systems of the L-cube are strongly normalizing.

4.18 Corollary
Deductions in a system on the L-cube are strongly normalizing.

Proof
The propositions-as-types map

$$|\,|: \text{L-cube} \to \lambda\text{-cube}$$

preserves reduction; moreover, the systems on the λ-cube are strongly normalizing. ■

In Leivant (1989) interesting use has been made of the propositions-as-types interpretation concerning the representation of data types.

The following example again shows the flexibility of the notion of GTS.

4.19 Example (Geuvers 1990)
The system of higher order logic in Church (1940) can be described by the following GTS

λHOL

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

That is **λHOL** is λω plus $\Box : \Delta$. The sound interpretation of **λPREDω** in **λHOL** is determined by the map given by

$$*^p \mapsto *$$
$$*^s \mapsto \Box$$
$$\Box^p \mapsto \Box$$
$$\Box^s \mapsto \Delta.$$

Geuvers (1990) proves that completeness holds for this interpretation.

Acknowledgements

The author wishes to thank Philips Research Laboratories at Eindhoven, Netherlands where Dr B. van Benthem Jutting explained to him several typed lambda calculi in the AUTOMATH family of languages of de Bruijn (1970). Also, thanks are due to Adriana Compagnoni, Maribel Fernandez and the paper's referee for pointing out some misprints, and especially to Erik Barendsen for tidying up the layout of the manuscript.

This paper is partly sponsored by the EC stimulation project 'Lambda calcul typé'. An earlier version appeared in the *Proceedings of the Third Italian Conference on Theoretical Computer Science* (Mantova, 1989), World Scientific, Singapore.

References

Barendregt, H. P. 1984. *The Lambda Calculus: Its Syntax and Semantics* (2nd Edn). North-Holland.

Barendregt, H. P. 1991. Lambda calculi with types. In S. Abramsky, D. Gabbai and T. Maibaum (editors), *Handbook of Logic in Computer Science*. Oxford University Press.

Barendregt, H. P. and van Leeuwen, M. 1985. Functional programming and the language TALE. In *Lecture Notes in Computer Science*, 224, pp. 122–208. Springer-Verlag.
Barendregt, H. P. and Hemerik, K. 1990. Types in lambda calculi and programming languages. In *Proc. European Symposium on Programming*, pp. 1–35, Copenhagen, Denmark (May).
Barendregt, H. P. and Dekkers, W. 1990. Typed lambda calculi.
Barendsen, E. 1989. *Representation of Logic, Data Types and Recursive Functions in Typed Lambda Calculi*. Masters thesis, University of Nijmegen, Netherlands.
Barendsen, E. and Geuvers, H. 1989. Conservativity of $\lambda\mathbf{P}$ over **PRED**. Manuscript. University of Nijmegen, Netherlands.
Berardi, S. 1988. Personal communication.
Berardi, S. 1990. *Type Dependence and Constructive Mathematics*. PhD thesis, Mathematical Institute, Torino, Italy.
de Bruijn, N. G. 1970. The mathematical language AUTOMATH, its usage and some of its extensions. In *Lecture Notes in Mathematics*, 125, pp. 29–61. Springer-Verlag.
de Bruijn, N. G. 1980. A survey of the AUTOMATH project. In J. R. Hindley and J. P. Seldin (editors), *To H. B. Curry: Essays on Combinatory logic, Lambda Calculus and Formalism*, pp. 580–606. Academic Press.
Cardelli, L. and Wegner, P. 1985. On understanding types, data abstraction and polymorphism. *ACM Comp. Surveys*, 17 (4): 471–522.
Church, A. 1940. A formulation of the simple theory of types. *J. Symbolic Logic*, 5: 56–68.
Coquand, Th. 1989. An introduction to type theory. To appear in A. R. Meyer (editor), *Proc. Ecole de Printemps du LITP*, Albi.
Coquand, Th. and Huet, G. 1988. The calculus of constructions. *Information and Computation*, 76: 95–120.
Curry, H. B. and Feys, R. 1958. *Combinatory logic*. North Holland.
van Daalen, D. 1980. *The Language Theory of AUTOMATH*. PhD. thesis, Technical University Eindhoven, Netherlands.
van Dalen, D. 1983. *Logic and Structure*. (2nd edn). Springer-Verlag.
Fujita, K. 1989. Relationship between logic and type system. Unpublished manuscript. Research Institute of Electrical Communication, Tohoku University, Japan.
Geuvers, H. 1988. *The Interpretation of Logics in Type Systems*. Master thesis, University of Nijmegen, Netherlands.
Geuvers, H. 1989. Theory of constructions is not conservative over higher order logic. Manuscript. University of Nijmegen, Netherlands.
Geuvers, H. 1990. Type systems for higher order logic. Manuscript. University of Nijmegen, Netherlands.
Geuvers, H. and Nederhof, M.-J. 1991. A modular proof of strong normalization for the theory of constructions. *Journal of Functional Programming* 1(2): 155–189.
Girard, J.-Y. 1972. *Interprétation Fonctionelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieur*. Thèse de Doctorat d'État, Université Paris VII, France.
Gordon, M. H., Milner, R. and Wadsworth, C. 1979. *Edinburgh LCF*: Lecture Notes in Computer Science, 78. Springer-Verlag.
Harper, R., Honsell F. and Plotkin, G. 1987. A framework for defining logics. In *Proc. 2nd Symp. Logic in Computer Science*, pp. 194–204. Ithaca, New York.
Howard, W. A. 1980. The formulae-as-types notion of construction. In J. R. Hindley and J. P. Seldin (editors), *To H. B. Curry: Essays on Combinatory logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press.
Leivant, D. 1989. Contracting proofs to programs. In: Odifreddi, P. (editor), *Logic and Computer Science*, pp. 279–327, Academic Press.
Longo, G. and Moggi, E. 1988. *Constructive Natural Deduction and its Modest Interpretation*. Report CMU-CS-88-131, Carnegie Mellon University, Pittsburgh, USA.
Martin-Löf, P. 1970. A construction of the provable wellorderings of the theory of species. Unpublished Manuscript. Mathematical Institute, University of Stockholm, Sweden.

Martin-Löf, P. 1984. *Intuitionistic Type Theory*. Bibliopolis.
Milner, R. 1984. A proposal for standard ML. In *Proc. 1984 ACM Symposium on LISP and Functional Programming*, pp. 184–197. Austin, Texas.
Mostowski, A. 1951. On the rules of proof in the pure functional calculus of first order. *J. Symbolic Logic*, 16: 107–111.
Peremans, W. 1949. Een opmerking over intuitionistische logica. Report ZW-16. Center for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam.
Prawitz, D. 1965. *Natural Deduction*. Almqvist and Wiksell.
Renardel de Lavalette, G. 1987. Strictness analysis for a language with polymorphic and recursive types (preprint). Department of Philosophy, Utrecht University, Netherlands.
Reynolds, J. 1974. Towards a theory of type structure. In *Proc. Colloque sur la Programmation.* In *Lecture Notes in Computer Science*, 19, pp. 408–425. Springer-Verlag.
Reynolds, J. 1985. Three approaches to type theory. In *Lecture Notes in Computer Science*, 185, pp. 145–146. Springer-Verlag.
Stenlund, S. 1972. Combinators, λ-terms and proof theory. D. Reidel.
Swaen, M. D. G. 1989. *Weak and Strong Sum-elimination in Institutionistic Type Theory*, PhD. thesis, University of Amsterdam, Netherlands.
Terlouw, J. 1988. Personal communication.
Turner, D. 1985. Miranda: A non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud (editor). *Functional Programming Languages and Computer Architecture. Lecture Notes in Computer Science*, 201, pp. 1–16. Springer-Verlag.
de Vrijer, R. 1975. Big trees in a λ-calculus with λ-expressions as types. In *Proc. Symposium on λ-calculus and computer science theory, Lecture Notes in Computer Science*, 37, pp. 252–271. Springer-Verlag.

Henk Barendregt, Faculty of Mathematics and Computer Science, Catholic University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands.