SLAC – PUB – 3626
April 1985
(T/E)

ON THE LOGICAL FORM OF MATHEMATICAL LANGUAGE*

CHRISTOFFER GEFWERT

Academy of Finland, Helsinki

and

Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

Submitted for Publications

* Work supported in part by the Department of Energy, contract DE-AC03-76SF00515 and by the Academy of Finland.

...there are ... certain limits to what
verbal explanations can do when it comes to
justifying axioms and rules of inference. In
the end, everybody must understand for himself.

Per Martin - Löf

1. Introduction

In this paper we are to outline the *logical form* of the language of mathematics based on a lecture given by Per Martin - Löf in Oxford 1975.[1] The language is an *extension* of the intuitionistic theory of types which is the subject of a series of papers by Martin-Löf.[2] The intuitionistic theory of types is itself an extension of Russell's doctrine of types which says that "(e)very propositional function $\phi(x)\ldots$ has, in addition to its range of truth, a range of significance, i.e. a range within which x must lie if $\phi(x)$ is to be a proposition at all, whether true or false. This is the first point in the theory of types; the second point is that ranges of significance form *types*, i.e. if x belongs to the range of significance of $\phi(x)$, then there is a class of objects, the *type* of x, all of which must also belong to the range of significance of $\phi(x)$, however ϕ may be varied; and the range of significance is always either a single type or a sum of several whole types."[3] As we said above: Martin-Löf's theory of types is an extension of the doctrine above.

One can trace the origins of this system, as Göran Sundholm points out,[4] on the technical side to Stephen Kleene's notion of realizability,[5] and Läuchli's abstract version thereof.[6] Furthermore, very important sources of inspiration were Gödel's *Dialectica* translation,[7] and W. A. Howard's notion of "formulae - as - type."[8] Other important sources are the natural deduction systems of Gentzen,[9] taken in conjunction with Dag Prawitz's reduction procedures.[10] These systems are a significant improvement on the Hilbert style axiomatic systems, such as Zermelo - Fraenkel set theory, as regards semantics. Still an important source of inspiration was provided by the AUTOMATH project of de Bruijn and his co-workers.[11] The relationship of this project (computer programming) to the intuitionistic theory of types has been explicitly dealt with by Martin -

Löf.[12] This paper is intended as a contribution in order to provide a ***philosophical investigation*** (person program) in accordance with the ***proposition - as - rules*** idea.[13]

In the foundational studies of mathematics we are still faced with the unresolved problems which the subject ran into the early part of this century. Among these are

- to decide the merit of intuitionist structure on classical modes of reasoning.

- to decide the merit of predicative strictures on impredicative modes of reasoning.

of which the second cuts across the first. More recently we are also faced with the problem how to decide the relative merits of the multiplying Babel of artificial languages put up as analyses of mathematical practice. There are for example:

CLASSICAL

Principia Mathematica

Zermelo's set theory and its various extensions by further axioms, like choice, regularity, replacement, and so on.

von Neumann - Bernays - Gödel set theory

Quine's set theories

Morse - Kelly set theory

Scott's theory of the cumulative hierarchy

various languages inspired by category theory

de Bruijn's project AUTOMATH

on the classical side, and

INTUITIONIST

Heything's intuitionistic logic

Kleene's and Vesley's axiomatic analysis

Bishop's numerical interpretation

Myhill's and Friedman's constructivist
set theories.

Martin - Löf's intuitionistic theory of types

Actzel's type theoretic interpretation of
constructivist set theory

Scott's project of constructivist validity

on the intuitionist side. To come to terms with these problems, we will have to explain such primitive mathematical concepts as

- set
- function
- natural number

- the natural number zero
- the successor of a natural number
- proposition
- propositional function, property or relation
- negation, conjunction, disjunction and implication
- universal and existential quantification
- identity

The explanation of defined concepts, like real numbers, Euclidean space, and category, is handled adequately within mathematics. Such concepts are reduced to others, more primitive ones, by explicit definitions. Evidently their explanation must be of a different kind. The aim of this paper is to contribute towards such a *philosophical* explanation.

2. The Logical Grammar of Mathematical Language

A philosophical explanation of mathematical language is given by a *Begriffschrift.* To engage in coding a *Begriffschrift* is to engage in writing a logical depth grammar. It is to engage in a philosophical investigation were the subject is a *participator.*[14] It is done in accordance with Wittgenstein's principle: the sense of a proposition can only be given once, i.e. the sense of a proposition cannot be expressed except by repeating that proposition.[15] For each fact there is necessarily only one proposition which answers to it. As Wittgenstein expressed it: "The rules of grammar cannot be justified by shewing that their application makes a representation agree with reality. For this justification would itself have to describe what is represented".[16] When we write a logical depth grammar we distinguish between a canonical part (syntax) providing the essential explanation and the informal part (semantics) providing the informal, verbalized, explanation. Informal explanations are used e.g. in different textbooks, but are reductible to a canonical form. One can say that a canonical form is a *criterion* for the use of the informal explanations vis-a-vis meaning, whereas the informal explanations are *symptoms* of there being a canonical form regulating their use. The different symptoms are *complementary* to each other vis-a-vis their use. And they are complementary precisely in virtue of the canonical form regulating them. Symptoms provide the *sufficient* conditions in order to engage in producing knowledge of facts (e.g. by computations or measurements), whereas criteria, or canonical forms, are the *necessary* conditions. What we do when engaging in writing a logical depth grammar is to *show* what makes it possible to produce knowledge of facts.[17] This is the essence of a person program, and for mathematical language it looks like what follows:

BEGRIFFSCHRIFT

$\overline{a} = x$	- x is the value (denotation) of a
	- a denotes x
$\overline{A} = X$	- X is the value (denotation) of A
	- A denotes X
$x \in X$	- x is an object of (belongs to, is an element of) the type (category, basic set) X

	- x is a proof of the proposition X
	- Theorem: X. Proof: x
X is a type	- X is a category (basic set)
	- X is a proposition
$\overline{a} \in \overline{A}$	- a is an $\overline{A}$ -valued function
	- a denotes an object (a proof) of the type (proposition) denoted by A
$\overline{A}$ is a type	- A is a type-valued (propositional) function
	- A denotes a type (proposition)
$\overline{A} \ni \overline{a} = \overline{b} \in \overline{B}$	- $a \equiv b$ (Curry, Bishop)
	- $a =_{def} b$
	- $a = b\,Df$ (Russell)
	- the $\overline{A}$-valued function a and the $\overline{B}$-valued function b are definitionally equal (have the same value)
	- a and b denote the same object (proof) of the type (proposition) denoted by A and B
$\overline{A} = \overline{B}$	- $A \equiv B$
	- $A =_{def} B$
	- $A = B\,Df$
	- the type-valued (propositional) functions A and B are definitionally equal

	- A and B denote the same type (proposition)
$\overline{u} \in \overline{A}$	- let u denote an arbitrary object (proof) of the type (proposition) denoted by A
	- assume A
	- the function which takes an arbitrary object of the type denoted by A into itself
0	- zero
$S(x)$	- x'
	- the successor of x
N	- ω
	- the type of natural numbers
$(\Pi u \in A)B(u)$	- $\prod\limits_{u \in A} B_u$
	- the (cartesian) product of the family of types $B(u)$, $u \in A$
	- the type of abstracts of $\overline{B(u)}$-valued functions of the $\overline{A}$-valued variable u
	- $(\forall u \in A)B(u)$
	- for all $u \in A$, $B(u)$
$A \rightarrow B$	- B^A

- $\mathcal{F}(A;B)$ (Bourbaki)
- $F(A,B)$ (Bishop)
- FAB (Curry)
- $(A)B$ (Schütte)
- (A,B) (Church)
- $(A|B)$ (Cantor)
- the type of functions from A to B
- $A \supset B$
- A implies B

$(\lambda u)b(u)$

- $\overset{\subset}{u}$ (Frege)
- $b(A)$ (Cantor)
- the abstract (Frege's *Wertverlauf*)
  of the function $b(u)$

- the object of type $(\Pi u \in A)B(u)$ $(A \rightarrow B)$
  which, when applied to an object x of the type
  denoted by A, yields as value the object that
  $b(u)$ denotes when x is assigned as value to the
  variable u

- the proof of $(\forall u \in A)B(u)$ which is obtained
  from the function $b(u)$ which takes and object x of the
  type denoted by A into a proof of the proposition denoted by
  $B(u)$ when x is assigned as value to the variable u

- the proof of $A \subset B$ which is obtained from

	the function $b(u)$ which takes a proof of the proposition denoted by A into a proof of the proposition denoted by B
$ap(u, w)$	- $w(u)$
	- functional application
	- universal instantiation
	- modus ponens
$(\Sigma u \in A)B(u)$	- $\sum_{u \in A} B_u$
	- the disjoint union (sum, coproduct) of the family of types $B(u), u \in A$
	- the type of pairs (x, y), where x is an object of the type denoted by A and y is an object of the type denoted by B
	- $(\exists u \in A)B(u)$
	- for some $u \in A$, $B(u)$
	- $\{u \in A \mid B(u)\}$
$A \times B$	- $(A . B)$ (Cantor)
	- the (cartesian) product of the two types A and B
	- the type of pairs (x, y), where x and y are objects of the type denoted by A and B, respectively
	- $A \& B$
	- $A \cdot B$
	- AB

	- A and B
	- (x, y)
	- the pair consisting of x and y
	- the proof of $(\exists u \in A)B(u)$ which is obtained from the object x of the type denoted by A and the proof y of the proposition denoted by $B(u)$ when x is assigned as value to the variable u
	- the proof of $A \& B$ which is obtained from the proofs x and y of the propositions denoted by A and B, respectively
$A + B$	- (A, B) (Cantor)
	- the disjoint union (sum, coproduct) of the two types A and B
	- $A \vee B$
	- A or B
$i(x) \quad j(y)$	- the canonical injection of the object $x\,(y)$ of the type denoted by $A(B)$ into $A + B$

3. Informal Explanations

We use

$$a,\ b,\ c,\ \ldots$$

for arbitrary object-valued functional expressions, and

$$A,\ B,\ C,\ \ldots$$

for arbitrary type-valued functional expressions. We use a parenthesis notation in conjunction with these letters and variables

$$u,\ v,\ w,\ \ldots$$

to indicate how they take arguments. Thus

$$a(u_1, \ldots, u_k)$$

will stand for an arbitrary object-valued functional expression whose variables are among $u_1, \ldots, u_k$. Similarly

$$A(u_1, \ldots, u_k)$$

will stand for an arbitrary type-valued functional expression whose variables are among $u_1, \ldots, u_k$. A different placing of $u_1, \ldots, u_k$ within the parentheses reflects a difference of input place within the functional expression. A single occurrence of a variable within the parentheses may stand for many linked argument places within the functional expression. Although one writes variables over functional expressions in two parts, these being a part for the variables, and a part for the expression over and above the variables, these two things cannot in any sense be separated in an actual functional expression. One may vary the functional expression while retaining the same structure of inputs, but not take away what is varied here, leaving the input structure so to speak on its own.

The following notations are used for arbitrary computations

$$\begin{array}{c} \bar{u}_1 = x_1 \quad \ldots \quad \bar{u}_k = x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1, \ldots, u_k)} = x \end{array} \qquad\qquad \begin{array}{c} \bar{u}_1 = x_1 \quad \ldots \quad \bar{u}_k = x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1, \ldots, u_k)} = X \end{array}$$

Here the upper lines show the assignments of arguments $x_1, \ldots, x_k$ to the variables $u_1, \ldots, u_k$ of the functional expression, and correspond to an indication of what inputs have been fed into what input holes of a machine.

The result of a computation!

$$\begin{array}{c} \overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1, \ldots, u_k)} = x \end{array}$$

is that

$$\overline{a(u_1, \ldots, u_k)} = x \textit{ for } \overline{u}_1 = x_1 \ \ldots, \ \overline{u}_k = x_k$$

This may be translated as follows

1. x is the *value* of $a(u_1, \ldots, u_k)$ for arguments $x_1, \ldots, x_k$ in the argument places $u_1, \ldots, u_k$ respectively, or when the variables $u_1, \ldots, u_k$ are assigned values $x_1, \ldots, x_k$ respectively.
2. x is the *output* of $a(u_1, \ldots, u_k)$ for inputs $x_1, \ldots, x_k$ in the input positions $u_1, \ldots, u_k$ respectively.
3. $a(u_1, \ldots, u_k)$ *denotes* x when $u_1, \ldots, u_k$ denote $x_1, \ldots, x_k$ respectively.

It is similar when we are concerned with the computation of a type-valued functional expression. We can now realize that the expressions *value*, *output* and *denotation*, as they have been used in mathematics, computer science and philosophy are used in a redundant way as far as meaning is concerned. Indeed, they are merely *symptoms* of there being a canonical form: a *criterion*.

Of course one may never proceed in a computation using in different places assignments of different object expressions as values to the same variable. For

A computation is not in the proper sense a proof of its result. It is something that produces its result. It is a process of which its result is the upshot, and not an argument for it. One can imagine a notation like the arrangement of addition and multiplication sums children are taught which one has no inclination to call the successive inference of propositions. The result of a computation is also not in the proper sense a relation between the functional expression, its inputs and its output, although Church for example refers to $\bar{a} = x$ as the name relation when a is a variable free functional expression. There is no harm in this terminology, provided one knows just what is meant by it.

The particular forms of object expression that we shall use to begin with are

$$O, \quad s(x), \quad (x, y), \quad i(x), \quad j(y), \quad r(x), \quad 1, \quad \ldots, \quad n, \quad (\lambda u) b(x_1, \ldots, x_k, u)$$

respectively. We shall see later why this is indeed what is essential to the notation. The use of the ordinary lambda notation is inadequate in this respect, unless it is supplemented somehow. For we cannot see from a lambda term what in it belongs to its form, and what to the constituents.

Because of the use of the last form of object expression, functional expressions give rise to forms of object expression. We shall see presently that, conversely, forms of object expression give rise to new functional expressions.

The particular forms of type expression that we shall use to begin with are

$$N, \quad N_0, \quad N_1, \quad \ldots, \quad N_k, \quad \ldots$$

Moreover, if $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k)$ are type-valued functional expressions, then

$$A(x_1, \ldots, x_k) + B(x_1, \ldots, x_k)$$

is a form of type expression. What is essential here is that from $A(x_1, \ldots, x_k)+ B(x_1, \ldots, x_k)$ we should be able to recover the sign $+$, the type-valued functional expressions $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k)$, and the object expressions $x_1, \ldots, x_k$ that are associated with the argument places in $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k)$ indicated by the variables $u_1, \ldots, u_k$ respectively.

Also, if $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k, u)$ are type-valued functional expressions, then both

$$(\Pi u \in A(x_1, \ldots, x_k)) B(x_1, \ldots, x_k, u)$$

and

$$(\Sigma u \in A(x_1, \ldots, x_k)) B(x_1, \ldots, x_k, u)$$

are forms of type expression. What is essential here is that from $(\Pi u \in A(x_1, \ldots, x_k)) B(x_1, \ldots, x_k, u)$ we should be able to recover the sign Π, the type-valued functional expressions $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k, u)$, the argument places in $B(u_1, \ldots, u_k, u)$ indicated by the variable u which are governed by the sign Π, and the object expressions $x_1, \ldots, x_k$ that are associated with the argument places in $A(u_1, \ldots, u_k)$ and $B(u_1, \ldots, u_k, u)$ indicated by the variables $u_1, \ldots, u_k$ respectively. It is similar with $(\Sigma u \in A(x_1, \ldots, x_k)) B(x_1, \ldots, x_k, u)$ save that the sign Σ takes the place of the sign Π.

When there is actually no argument place $B(u_1, \ldots, u_k, u)$ indicated by u, so that this type-valued functional expression may also be written $B(u_1, \ldots, u_k)$, then these forms will be written synonymously

$$A(x_1, \ldots, x_k) \rightarrow B(x_1, \ldots, x_k)$$

and

$$A(x_1, \dots, x_k) \times B(x_1, \dots, x_k)$$

respectively.

Finally, a type-valued functional expression $A(u_1, \dots, u_k)$ gives rise to a form

$$I_{A(x_1, \dots, x_k)}(x, y)$$

of type expressions. What is essential here is that from $I_{A(x_1, \dots, x_k)}(x, y)$ we may recover the sign I, the type-valued functional expression $A(u_1, \dots, u_k)$, and the object expressions $x_1, \dots, x_k, x, y$ where $x_1, \dots, x_k$ are associated with the argument places of $A(u_1, \dots, u_k)$ indicated by the variables $u_1, \dots, u_k$.

4. The Formal Part

Now we deal with functional expressions employed in the portion of the language to be dealt with first. A functional expression, whether object-valued or type-valued, is simple or composite, *and a simple functional expression is primitive or defined. It is primitive or defined in virtue of the kind of computation rule laid down for it.* We treat primitive functional expressions first. If $f(x_1, \dots, x_k)$ is a form of object expression, then $f(u_1, \dots, u_k)$ is a primitive object-valued functional expression, which operates according to the computation rule

$$\frac{\overline{u_1} = x_1 \quad \dots \quad \overline{u_k} = x_k}{\overline{f(u_1, \dots, u_k)} = f(x_1, \dots, x_k)}$$

Thus, for example, as special cases of this rule we have

$$\overline{0} = 0 \qquad \frac{\overline{u} = x}{\overline{s(u)} = s(x)}$$

The primitive object-valued functional expression $f(u_1, \ldots, u_k)$ must not be confused with the form $f(x_1, \ldots, x_k)$ of object expression, since *it is stipulated that something is a form of object expression, and not stipulated, but to be understood, that something is a functional expression.* One could mark this in the notation for the functional expression itself, by for example using $(S)u$ for the primitive functional expression arising from the form $s(x)$. However, it will be (in a precise sense) always clear from the context whether the functional expression or the form is meant.

Similarily, a form $F(x_1, \ldots, x_k)$ of type expression gives rise to a primitive type-valued functional expression $F(u_1, \ldots, u_k)$ which operates according to the computation rule

$$\frac{\overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k}{\overline{F(u_1, \ldots, u_k)} = F(x_1, \ldots, x_k)}$$

Next we treat defined functional expressions. That is, we engage in giving the *formation - rules.* In the portion of the language developed here, there are in fact no defined type-valued functional expressions, so we treat only defined object-valued functional expressions. The argument places of a defined functional expression always include one called the *principal* argument place of the functional expression, the remainder being called *subordinate.* The distinction between these is that the computation of a defined functional expression always proceeds according to the form of the argument put into the principal argument place, while it is always uniform in the remaining arguments. That is, for the computation of such a function, we look first to the form of the principal argument to see how it is to be computed, while the forms of the other arguments become critical only at later stages of the computation. In this sense, what is in the principal argument place, or at least its form, take precedence over what is in the subordinate argument places, and indeed also the constituents of the principal argument.

As a matter of convention, in writing

$$f(u_1, \ldots, u_k, u)$$

it is always supposed that the principal argument place is that indicated by the variable u, while the subordinate argument places are those indicated by $u_1, \ldots, u_k$. Furthermore, instead of writing

$$\overline{f(u_1, \ldots, u_k, u)} = y \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_k = x_k, \ \overline{u} = x$$

in giving the computation rules for such functional expressions, we write simply

$$f(x_1, \ldots, x_k, x) = y$$

The rules by which defined object-valued functional expressions arise are given by the following list of definitional schemes.

4.1 N-Scheme or Recursion

From object-valued functional expressions $a(u_1, \ldots, u_k)$ and $b(u_1, \ldots, u_k, u, v)$ there arises an object-valued functional expression $f(u_1, \ldots, u_k, u)$ defined by the rules.

$$\frac{\begin{array}{c}\overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1, \ldots, u_k)} = x\end{array}}{f(x_1, \ldots, x_k, 0) = x}$$

$$\frac{f(x_1, \ldots, x_k, x) = y \qquad \begin{array}{c}\overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k \quad \overline{u} = x \quad \overline{v} = y \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{b(u_1, \ldots, u_k, u, v)} = z\end{array}}{f(x_1, \ldots, x_k, s(x)) = z}$$

We may sometimes write

$$(Rv, w)\,(a(u_1, \ldots, u_k),\, b(u_1, \ldots, u_k, v, w)\,, u)$$

for the functional expression so defined. Whatever the notation, it is essential that we may recover enough information to know how it is computed.

4.2 Π*-Scheme*

$ap(u, w)$ is an object-valued functional expression in two variables defined by the scheme

$$\frac{\begin{array}{c}\overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k \quad \overline{u} = x \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{b(u_1, \ldots, u_k, u)} = y\end{array}}{ap(x,\, (\lambda u)\, b\,(x_1, \ldots, x_k,\, u)) = y}$$

This is why what we previously stated to be essential in the object expression $(\lambda u)\, b(x_1, \ldots, x_k, u)$ is indeed essential. For, in order to work out the value of $ap(u, w)$ for $\overline{u} = x$ and $\overline{w} = (\lambda u)\, b(x_1, \ldots, x_k, u)$, we must able to recover from $(\lambda u)\, b(x_1, \ldots, x_k, u)$, the functional expression $b(u_1, \ldots, u_k, u)$, the argument places in it to be filled by the subordinate argument x, namely those shown by u, the object expressions $x_1, \ldots, x_k$ and the variable places in $b(u_1, \ldots, u_k, u)$ which they are to fill, namely those indicated by the variables $u_1, \ldots, u_k$.

4.3 Σ-Scheme

Given an object-valued functional expression $c(u_1, \ldots, u_k, v, w)$ we may define an object-valued functional expression $f(u_1, \ldots, u_k, u)$ by the scheme

$$\dfrac{\begin{array}{c}\overline{u}_1 = x_1 \quad \ldots \quad \overline{u}_k = x_k \quad \overline{v} = x \quad \overline{w} = y \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{c(u_1, \ldots, u_k, v, w)} = z\end{array}}{f(x_1, \ldots, x_k, (x, y)) = z}$$

We may sometimes write

$$(Ev, w)\,(c(u_1, \ldots, u_k, v, w)\,, u)$$

for the functional expression defined in this way from $c(u_1, \ldots, u_k, v, w)$.

4.4 +-Scheme

Given object-valued functional expressions $a(u_1, \dots, u_k, v)$ and $b(u_1, \dots, u_k, w)$, we may define an object-valued functional expression $f(u_1, \dots, u_k, u)$ by the scheme

$$\begin{array}{c} \overline{u}_1 = x_1 \quad \dots \quad \overline{u}_k = x_k \quad \overline{v} = x \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1, \dots, u_k, v)} = z \\ \hline f(x_1, \dots, x_k, i(x)) = z \end{array}$$

$$\begin{array}{c} \overline{u}_1 = x_1 \quad \dots \quad \overline{u}_k = x_k \quad \overline{w} = y \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{b(u_1, \dots, u_k, w)} = z \\ \hline f(x_1, \dots, x_k, j(y)) = z \end{array}$$

We may sometimes use the notation

$$(Dv, w)(a(u_1, \dots, u_k, v), b(u_1, \dots, u_k, w), u)$$

for $f(u_1, \dots, u_k, u)$.

4.5 I-Scheme

If $c(u_1, \dots, u_k, u)$ is an object-valued functional expression, the we may define an object-valued functional expression $f(u_1, \dots, u_k, u, v, w,)$ by the scheme

$$\begin{array}{c} \overline{u}_1 = x_1 \quad \dots \quad \overline{u}_k = x_k \quad \overline{u} = x \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{c(u_1, \dots, u_k, u)} = z \\ \hline f(x_1, \dots, x_k, x, x, r(x)) = z \end{array}$$

4.6 N_n-Scheme

Given object-valued functional expressions

$$a_1(u_1,\ldots,u_k),\ldots a_n(u_1,\ldots,u_k) \quad ,$$

we may define an object-valued functional expression $f(u_1,\ldots,u_k,u)$ by the scheme

$$\frac{\begin{array}{c}\overline{u_1}=x_1 \qquad \cdots \qquad \overline{u_k}=x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a_m(u_1,\ldots,u_k)}=x\end{array}}{f(x_1,\ldots,x_k,m)=x} \qquad\qquad m=1,\ldots,n$$

Note that, when $n = 0$, the scheme is empty. Hence the computation of the defined functional expression is in such a case always trivial, in the sense that there is no form of principal argument at which the computation of the functional expression can begin.

There remains the notion of a composite functional expression. Firstly, if $a_1,\ldots,a_k$ and $b(u_1,\ldots,u_k)$ are object-valued functional expressions, then $b(a_1,\ldots,a_k)$ is an object-valued functional expression, which operates according to the rule

$$\frac{\overline{a_1}=x_1 \quad \cdots \quad \overline{a_k}=x_k \qquad \begin{array}{c}\overline{u_1}=x_1 \quad \cdots \quad \overline{u_k}=x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{b(u_1,\ldots,u_k)}=y\end{array}}{\overline{b(a_1\ldots,a_k)}=y}$$

Secondly, if $a_1,\ldots a_k$ are object-valued functional expressions and $A(u_1,\ldots,u_k)$ is a type-valued functional expression, then $A(a_1,\ldots,a_k)$ is a type-valued functional expression, which operates according to the rule

$$\dfrac{\bar{a}_1 = x_1 \quad \dots \quad \bar{a}_k = x_k \qquad \begin{array}{c} \bar{u}_1 = x_1 \quad \dots \quad \bar{u}_k = x_k \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\dots,u_k)} = X \end{array}}{\overline{A(a_1 \dots, a_k)} = X}$$

5. On The General Method Of Commutability

Given functional expressions $a_1, \dots, a_k$ and computations of $\bar{a}_1 = x_1, \dots, \bar{a}_k = x_k$, we may transform a computation of $\overline{a(a_1,\dots,a_k)} = x$ into a computation of $\overline{a(u_1,\dots,u_k)} = x$ for $\bar{u}_1 = x_1, \dots, \bar{u}_k = x_k$, and, similarly, a computation of $A(a_1 \dots, a_k)$ into one of $\overline{A(u_1,\dots,u_k)}\,\bar{u}_1 = x_1, \dots, \bar{u}_k = x_k$. The computation of $(a_1 \dots, a_k) = x$ consists of computations of certain values for the component functional expressions of $(a_1 \dots, a_k)$ for various arguments. It is now merely a matter of reordering the applications of the rules for the evaluation of composite functions, for we can certainly extract computations of the values of $a_1 \dots, a_k$. And, these values will be $x_1, \dots, x_k$ respectively, since a functional expression determines its value. What remains will then be sufficient to put together a computation of $\overline{a(u_1,\dots,u_k)} = x$ for $\bar{u}_1 = x_1, \dots, \bar{u}_k = x_k$.

The commutability of composition with evaluation is not a mathematical proposition that one gives a mathematical proof. For in the first place, unless one regards the language meta-mathematically, within some more extensive language, neither the object expressions that are the arguments in a computation, nor the functional expression one computes, nor the object or type expression one gets as its value, are mathematical objects that can figure in a mathematical proposition. Secondly, the result of a computation is not a mathematical proposition that one can combine with others by means of the usual logical operations to get the commutability thesis, unless, once more, one regards the language metamathematically. Rather, what we are concerned with is not a theorem and its proof, but the statement of a simple mechanical task, and the explanation of how to do it.

6. The Non-formal Part

The non-formal (or teleological) part of the language deals with ***sentences*** of the forms

$x \in X$

X is a type

$\bar{a} \in \bar{A}$

$\bar{A}$ is a type

$\bar{A} \ni \bar{a} = \bar{b} \ni \bar{B}$

$\bar{A} = \bar{B}$

These may be read informally as follows. First

x is an object of (belongs to, is an element of) the type (category, basic set) X

x is a (canonical) proof of the proposition X

Theorem: X. proof: x

Second

X is a category (basic set)

X is a proposition

Now we can understand Wittgenstein's comment that "(g)rammar is a 'theory of logical types'."[18] What Wittgenstein is alluding to is the non - formal part of the depth grammar of mathematical Language. Indeed, a philosophical investigation provides an extension of Russell's doctrine of types, according to which a type in the range of significance of a propositional function, because, "... every function, and thus, in particular, every propositional function, will indeed have a type as its domain. A type is defined by prescribing what we have to do in order to construct an object of that type. This is almost verbatim the definition of the notion of set given by Bishop".[19] Furthermore, this is the first place at which a redundancy of a rather striking kind in the informal language is eliminated from the formal one. The informal notion of ***proposition*** and that of ***set*** (or, at least, one constituent of that notion) are the same, and their separation within informal language, and the traditional formal languages, constitutes a redundancy. The true reason why these notions are the same is not the formal similarity of the rules in the traditional languages governing the generation of objects of given types, and of proofs of given propositions. That similarity was what struck Curry and Feys in the domain of positive implicational logic,[20] and Howard showed how to extend as for as intuitionistic first order arithmetic.[21] Rather it is that the explanations of what is expressed by

x is an element of X

and

x is a proof of X

which we shall give later on, are the same.

Third

a is an $\overline{A}$-valued function the value of a is an object (a proof) of the type (proposition) which is the value of A

Fourth

A is a type-valued function, that is, a function whose values are types

A is a propositional function, that is, a function whose values are propositions

Fifth

$$a \equiv b, \quad a =_{\text{def}} b, \quad a = b\,Df$$

the $\overline{A}$-valued function a and the $\overline{B}$-valued function b are definitionally equal, or equivalent.[22]

Sixth

$$A \equiv B, \quad A =_{\text{def}} B, \quad A = B\,Df$$

the type-valued functions A and B are definitionally equal, or equivalent.

A sentence in the Language is an expression of one of the above six forms. That is to say: the depth grammar of the Language only requires expressions of these six forms. Note that the constituents of the first two forms of sentence are object or type expressions, whereas in the last four forms they are functional expressions. Accordingly, while the first two forms are of rather the same kind as the forms of object and type expression, the latter four are essentially different.

The sentences of the language will express certain tasks, or objectives, which we must explain before explaining how the rules of the language are understood. However, that is not what is essential to a sentence, rather it is that it is one of the forms shown above.

A sentence of one of the latter four forms that contains variables will always be derived from assumptions which assign ranges to these variables, and to indicate an arbitrary such derivation, we use the notations

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \qquad \cdots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1,\ldots,u_k)} \in \overline{A(u_1,\ldots,u_k)} \end{array}$$

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \qquad \cdots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\ldots,u_k)} \text{ is a type} \end{array}$$

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \qquad \cdots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\ldots,u_k)} \ni \overline{a(u_1,\ldots,u_k)} \quad = \quad \overline{b(u_1,\ldots,u_k)} \in \overline{B(u_1,\ldots,u_k)} \end{array}$$

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \qquad \cdots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\ldots,u_k)} = \overline{B(u_1,\ldots,u_k)} \end{array}$$

The upper line

$$\overline{u}_1 \in \overline{A}_1 \qquad \cdots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})}$$

shows assumptions including those with which the derivation begins. The variables occurring in such a figure serve to link together and distinguish the various argument places of the functional expressions occurring in the figure, through their identity and difference. (They link and distinguish not only the argument places in a single functional expressions throughout the figure.) This might be achieved by many other devices than the use of variables. That however is what is essential to the use of variables here. The above figures are rendered into informal terms as follows

1. $a(u_1, \ldots, u_k)$ is an $\overline{A(u_1, \ldots, u_k)}$-valued function of the $\overline{A}_1$-valued variable u_1, ..., the $\overline{A_k(u_1, \ldots, u_{k-1})}$-valued variable u_k

2. the value of $a(u_1, \ldots, u_k)$ is an object of the type which is the value of $A(u_1, \ldots, u_k)$ when u_1 is assigned a value from the type which is the value of $A(u_1, \ldots, u_k)$ is assigned a value from the type which is then the value of $A_k(u_1, \ldots, u_{k-1})$

and similarly in the other three cases.

What is expressed by a sentence of one of the latter four forms which contains variables $u_1, \ldots, u_k$ is always explained relative to assumptions $\overline{u}_1 \in \overline{A}_1, \ldots, \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}$. Accordingly it would be more suitable, so far as the depth grammar is concerned, to recast the language so that we are concerned with sentences of the forms

$$\overline{a(u_1, \ldots, u_k)} \in \overline{A(u_1, \ldots, u_k)} \text{ for } \overline{u}_1 \in \overline{A}_1, \ \ldots, \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}$$

etc.

rather than those displayed above. This might with justice be called a *sequent formulation* of the Language, while the one we are giving might be called its natural deduction formulation, by analogy with the Gentzen systems.[23] The natural deduction formulation is preferred here for its compactness and its greater proximity to the informal languages.

We shall write

$\overline{a} = x \in X$ for $\overline{a} = x$ and $x \in X$, and

$x \in X = \overline{A}$ for $x \in X$ and $\overline{A} = X$.

Furthermore a definitional equality $\overline{A} \ni \overline{a} = \overline{b} \in \overline{A}$ in which the outer terms are the same will be written synonymously $\overline{a} = \overline{b}$ $(\overline{A})$.

6.1 The N-Rule Family

$$O \in N \qquad \frac{x \in N}{s(x) \in N} \qquad \text{N is a type}$$

$$\overline{O} \in \overline{N} \qquad \frac{\overline{u} \in \overline{N}}{\overline{s(u)} \in \overline{N}} \qquad \overline{N} \text{ is a type}$$

The fourth and fifth rules here will be called the ***introductory rules for*** *N*. Furthermore, if

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{a(u_1, \ldots, u_k)} \in \overline{A(u_1, \ldots, u_k, 0)}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \ \overline{u} \in \overline{N} \ \overline{v} \in \overline{A(u_1, \ldots, u_k, u)}}{\overline{b(u_1, \ldots, u_k, u, v)} \in \overline{A(u_1, \ldots, u_k, s(u))}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \ \overline{u} \in \overline{N}}{\overline{A(u_1, \ldots, u_k, u)} \text{ is a type}}$$

and $f(u_1, \ldots, u_k, u)$ is defined from $a(u_1, \ldots, u_k)$ and $b(u_1, \ldots, u_k, u)$ by the N-scheme, then

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{N}}{\overline{f(u_1, \ldots, u_k, u)} \in \overline{A(u_1, \ldots, u_k, u)}}$$

which is the *elimination rule for N*, and

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})}}{\overline{A(u_1,\ldots,u_k,0)} \ni \overline{f(u_1,\ldots,u_k,0)} = \overline{a(u_1,\ldots,u_k)} \in \overline{A(u_1,\ldots,u_k,0)}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{u} \in \overline{N}}{\begin{array}{c}\overline{A(u_1,\ldots,u_k,\,s(u))} \ni \overline{f(u_1,\ldots,u_k,\,s(u))} \\ = \overline{b(u_1,\ldots,u_k,\,u,\,f(u_1,\ldots,u_k,u))} \in \overline{A(u_1,\ldots,u_k,\,s(u))}\end{array}}$$

The forms of object and type expression treated in these rules may be translated in the informal language as follows

0	–	zero
$s(x)$	–	x',
	–	the successor of x
N	–	ω,
	–	the type of natural numbers

6.2 The Π*-Rule Family*

If

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\ldots,u_k)} \text{ is a type}\end{array}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\ldots,u_k)}}{\overline{b(u_1,\ldots,u_k,u)} \in \overline{B(u_1,\ldots,u_k,u)}}$$

then

$$\frac{x_1 \in X_1 = \overline{A}_1 \quad \ldots \quad x_k \in X_k = \overline{A_k(u_1, \ldots, u_k)} \text{ for } \overline{u}_1 = x, \ldots, \overline{u}_{k-1} = x_{k-1}}{(\lambda u)\, b(x_1, \ldots, x_k, u) \in (\Pi u \in A(x_1, \ldots, x_k))\, B(x_1, \ldots, x_k, u)}$$

which is called the ***rule of abstraction***, and

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{(\lambda u) b(u_1, \ldots, u_k, u)} \in \overline{\Pi u \in A(u_1, \ldots, u_k))\, B(u_1, \ldots, u_k, u)}}$$

which is called the Π-***introduction rule.***

Furthermore, if

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1, \ldots, u_k)} \text{ is a type} \end{array}$$

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_k)} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{B(u_1, \ldots, u_k, u)} \text{ is a type} \end{array}$$

then

$$\frac{\begin{array}{r} x_1 \in X_1 = \overline{A}_1 \quad \ldots \quad x_k \in X_k = \overline{A_k(u_1, \ldots, u_{k-1})} \text{ for } \overline{u}_1 = x, \\ \ldots, \overline{u}_{k-1} = x_{k-1} \end{array}}{(\Pi u \in A(x_1, \ldots, x_k))\, B(x_1, \ldots, x_k, u) \text{ is a type}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{(\Pi u \in A(u_1, \ldots, u_k))\, B(u_1, \ldots, u_k, u)} \text{ is a type}}$$

Also

$$\frac{\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_k)} \\ \overline{v} \in \overline{(\Pi u \in A(u_1, \ldots, u_k))\, B(u_1, \ldots, u_k, u)} \end{array}}{\overline{ap(u, v)} \in \overline{B(u_1, \ldots, u_k, u)}}$$

and

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\dots,u_k)}}{\overline{B(u_1,\dots,u_k,u)} \ni \overline{ap(u,(\lambda u)b(u_1,\dots,u_k,u))} \\ = \overline{b(u_1,\dots,u_k,u)} \in \overline{B(u_1,\dots,u_k,u)}}$$

provided

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\dots,u_k)} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{b(u_1,\dots,u_k,u)} \in \overline{B(u_1,\dots,u_k,u)} \end{array}$$

This last is called the ***rule of*** λ***-conversion.***

The forms of object and type expression treated in these rules may be translated into the informal language as follows (we suppress all variables that are not critical in the form we are considering).

$(\Pi u \in A)B(u)$	$- \prod_{u \in A} B_u$
	– the (cartesian) product of the family of types B_u, $u \in A$
	– the type of abstracts of $B(u)$-valued functions of the A-valued variable u
	– $(\forall u \in A)B(u)$
	– for all $u \in A$, $B(u)$

Here is another place where a redundancy in the informal language is eliminated from the formal one. Again, the reason why the separation of ***cartesian products*** from ***universal quantification*** is a redundancy is not merely that the rules governing them are formally similar, but that the explanation of these rules is the same.

Furthermore, in the case when $B(u)$ does not depend on u, when we write $A \rightarrow B$ synonymously with $(\Pi u \in A)B(u)$, the following are informal translations of $(\Pi u \in A)B(u)$

$$B^A, \quad \mathcal{F}(A,B), \quad F(A,B), \quad FAB, \quad (A)B, \quad (A,B), \quad (A|B)$$

the type of functions from A to B

$$A \supset B$$

A implies B

Here is another redundancy in the informal language and the traditional logical systems, namely the separation not just between the notions of cartesian product and universal quantification, but also between these and the ***notion of implication***.

$(\lambda\, u)b(u)$ may be translated by

- $\underset{u}{\supset} b(u)$ in Frege's notation
- $b(A)$ in Cantor's notation
- the abstract (Frege's *Wertverlauf*) of the function $b(u)$
- the object of type $(\Pi u \in A)B(u)$ (resp. $A \rightarrow B$) which, when applied to an object x of the type which is the value of A, yields the value of $b(u)$ when u is assigned the valued x

- the proof of $(\forall u \in A)B(u)$ which is obtained from the function $b(u)$ which takes an object x of the type which is the value of A into a proof of the proposition that is the value of $B(u)$ when u is assigned the value x

- the proof of $A \supset B$ which is obtained from the function $b(u)$ which takes a proof x of the proposition which is the value of A into a proof of the proposition which is the value of B

Although in ordinary mathematics one does not distinguish between functions and their abstracts, that is, functions proper and functions as objects, the distinction is vital in our analysis of the ordinary practice. The notion of function as object cannot be understood unless we already have the notion of function in the proper sense. It would be futile to try to dispense with the latter notion, not least because application at any rate can only be understood as a function in this prior sense.

The defined functional expression $ap(u, w)$ allows us to translate the informal notation $w(u)$ for functional application, and also universal instantiation and modus ponens. The paranthesis notation $w(u)$ for functional application is in full accordance with the confusion in ordinary mathematics between the two notions of function.

6.3 The Σ-Rule Family

If

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{A(u_1, \ldots, u_k)} \text{ is a type}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_{k-1})}}{\overline{B(u_1, \ldots, u_k, u)} \text{ is a type}}$$

then

$$\frac{\begin{array}{l} x_1 \in X_1 = \overline{A}_1 \\ \quad\vdots \\ x_k \in X_k = \overline{A_k(u_1,\ldots,u_{k-1})} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_{k-1} = x_{k-1} \\ x \in X = \overline{A(u_1,\ldots,u_k)} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_k = x_k \\ y \in Y = \overline{B(u_1,\ldots,u_k,u)} \text{ for } \overline{u}_1 = x_k, \overline{u} = x \end{array}}{(x,y) \in (\Sigma u \in A(x_1,\ldots,x_k))B(x_1,\ldots,x_k,u)}$$

$$\frac{\begin{array}{r} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\ldots,u_k)} \\ \overline{v} \in \overline{B(u_1,\ldots,u_k,u)} \end{array}}{\overline{(u,v)} \in \overline{(\Sigma u \in A(u_1,\ldots,u_k))B(u_1,\ldots,u_k,u)}}$$

which is the Σ-*introduction rule*, and

$$\frac{\begin{array}{r} x_1 \in X_1 = \overline{A}_1 \quad \ldots \quad x_k \in X_k = \overline{A_k(u_1,\ldots,u_{k-1})} \text{ for } \overline{u}_1 = x, \\ \ldots, \overline{u}_{k-1} = x_{k-1} \end{array}}{(\Sigma u \in A(x_1,\ldots,x_k))B(x_1,\ldots,x_k,u) \text{ is a type}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})}}{\overline{(\Sigma u \in A(u_1,\ldots,u_k))B(u_1,\ldots,u_k,u)} \text{ is a type}}$$

Furthermore, if

$$\frac{\begin{array}{r} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\ldots,u_k)} \\ \overline{v} \in \overline{B(u_1,\ldots,u_k,u)} \end{array}}{\overline{c(u_1,\ldots,u_k,u,v)} \in \overline{C(u_1,\ldots,u_k,(u,v))}}$$

$$\frac{\begin{array}{l} \overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \overline{w} \in \overline{(\Sigma u \in A(u_1,\ldots,u_k))B(u_1,\ldots,u_k,u)} \end{array}}{\overline{C(u_1,\ldots,u_k,w)} \text{ is a type}}$$

and $f(u_1, \ldots, u_k, w)$ is defined from $c(u_1, \ldots, u_k, u, v)$ by the Σ-scheme, then

$$\frac{\begin{array}{c} \bar{u}_1 \in \overline{A}_1 \quad \ldots \quad \bar{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \bar{w} \in \overline{(\Sigma u \in A(u_1, \ldots, u_k))B(u_1, \ldots, u_k, u)} \end{array}}{\overline{f(u_1, \ldots, u_k, w)} \in \overline{C(u_1, \ldots, u_k, w)}}$$

which is the Σ-*elimination rule*, and

$$\begin{array}{l} \bar{u}_1 \in \overline{A}_1 \quad \ldots \quad \bar{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \bar{u} \in \overline{A(u_1, \ldots, u_k)} \\ \hfill \bar{v} \in \overline{B(u_1, \ldots, u_k, u)} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{C(u_1, \ldots, u_k, (u,v))} \ni \overline{f(u_1, \ldots, u_k, (u,v))} \\ \qquad\qquad = \overline{c(u_1, \ldots, u_k, u, v)} \in \overline{C(u_1, \ldots, u_k, (u,v))} \end{array}$$

The forms of object and type expression involved in these rules may be translated into informal terms as follows

- $\sum_{u \in A} B_u$
- the disjoint union (sum, coproduct) of the family of types B_u, $u \in A$
- the type of pairs (x, y) where x is an object of the type which is the value of A and y is an object of the type which is the value of $B(u)$ when u is assigned the value x
- $(\exists u \in A)B(u)$
- for some $u \in A$, $B(u)$

Similar remarks to those about redundancies eliminated by the rules apply here. There is one further notion of informal mathematics synonymous with the notions of disjoint union and existential quantification, namely the notion of ***such that***

- $\{u \in A \mid B(u)\}$

- the type of all $u \in A$ such that $B(u)$

The notion of set used in ordinary mathematics is really a mixture of notions, namely of property and type, $B(u)$ and $(\Sigma u \in A)B(u)$ respectively. These must be sharply distinguished in our analysis of the ordinary practice. (There are still other elements to be discerned within the informal notion.) An object of the type $(\Sigma u \in A)B(u)$ is an object x of the type which is the value of A *together with* a proof y of the proposition which is the value of $B(u)$ when u is assigned the value x, not merely, so to speak, the object x just happening to satisfy $B(u)$. This way of handling the notion of comprehension, or separation, is quite crucial throughout mathematics. For an example, which Myhill discusses, we may take the definition of the inverse function on the type of real numbers different from zero $\{u \in R \mid u \neq 0\}$. For the function depends not only on a given real number, but essentially on the proof that it is different from zero.

In the case when $B(u)$ does not depend on u, so that we write $A \times B$ synonymously with $(\Sigma u \in A)B(u)$, the following may be taken as informal translations

- $(A \,.\, B)$ in Cantor's notation

- the cartesian product of the types A and B

- the type of pairs (x, y) where x and y are objects of the types which are the values of A and B, respectively

- $A \,\&\, B$, $\quad A\ B$, $\quad A \,.\, B$, $\quad AB$, $\quad A$ and B

(x, y) may be translated as follows

- $\langle x, y \rangle$

- the pair consisting of x and y

- the proof of $(\exists u \in A)B(u)$ which is obtained from the object x of the type which is the value of A and the proof y of the proposition which is the value of $B(u)$ when u is the assigned the value x

- the proof of $A\&\ B$ which is obtained from the proofs x and y of the propositions which are the values of A and B respectively

6.4 The +-Rule Family

If

$$\begin{array}{c} \bar{u}_1 \in \bar{A}_1 \quad \cdots \quad \bar{u}_k \in \overline{A_k(u_1,\ldots,u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1,\ldots,u_k)} \text{ is a type} \end{array}$$

then

$$\begin{array}{l} x_1 \in X_1 = \bar{A}_1 \\ \quad\vdots \\ x_k \in X_k = \overline{A_k(u_1,\ldots,u_{k-1})} \text{ for } \bar{u}_1 = x_1, \ldots, \bar{u}_{k-1} = x_{k-1} \\ x \in X = \overline{A(u_1,\ldots,u_k)} \text{ for } \bar{u}_1 = x_1, \ldots, \bar{u}_k = x_k \\ \hline \qquad\qquad i(x) \in A(x_1,\ldots,x_k) + B(x_1,\ldots,x_k) \end{array}$$

$$\begin{array}{l} x_1 \in X_1 = \bar{A}_1 \\ \quad\vdots \\ x_k \in X_k = \overline{A_k(u_1,\ldots,u_{k-1})} \text{ for } \bar{u}_1 = x_1, \ldots, \bar{u}_{k-1} = x_{k-1} \\ y \in Y = \overline{B(u_1,\ldots,u_k)} \text{ for } \bar{u}_1 = x_1, \ldots, \bar{u}_k = x_k \\ \hline \qquad\qquad j(y) \in A(x_1,\ldots,x_k) + B(x_1,\ldots,x_k) \end{array}$$

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\dots,u_k)} \\ \hdashline \overline{i(u)} \in \overline{A(u_1,\dots,u_k) + B(u_1,\dots,u_k)}\end{array}$$

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{v} \in \overline{B(u_1,\dots,u_k)} \\ \hdashline \overline{j(v)} \in \overline{A(u_1,\dots,u_k) + B(u_1,\dots,u_k)}\end{array}$$

of which the last two are the +-*introduction rules*, and

$$\frac{x_1 \in X_1 = \overline{A}_1 \quad \dots \quad x_k \in X_k = \overline{A_k(u_1,\dots,u_k)} \text{ for } \overline{u}_1 = x, \dots, \overline{u}_{k-1} = x_{k-1}}{A(x_1,\dots,x_k) + B(x_1,\dots,x_k) \text{ is a type}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})}}{\overline{A(u_1,\dots,u_k) + B(u_1,\dots,u_k)} \text{ is a type}}$$

Furthermore, if

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\dots,u_k)} \\ \hdashline \overline{a(u_1,\dots,u_k,u)} \in \overline{C(u_1,\dots,u_k,i(u))}\end{array}$$

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \quad \overline{v} \in \overline{B(u_1,\dots,u_k)} \\ \hdashline \overline{b(u_1,\dots,u_k,v)} \in \overline{C(u_1,\dots,u_k,j(v))}\end{array}$$

$$\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \qquad \dots \qquad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \\ \overline{w} \in \overline{A(u_1,\dots,u_k) + B(u_1,\dots,u_k)} \\ \hdashline \overline{C(u_1,\dots,u_k,w)} \text{ is a type}\end{array}$$

and $f(u_1,\dots,u_k,w)$ is defined from $a(u_1,\dots,u_k,u)$ and $b(u_1,\dots,u_k,v)$ as in the +- *scheme*, then

$$\frac{\begin{array}{c}\overline{u}_1 \in \overline{A}_1 \qquad \dots \qquad \overline{u}_k \in \overline{A_k(u_1,\dots,u_{k-1})} \\ \overline{w} \in \overline{A(u_1,\dots,u_k) + B(u_1,\dots,u_k)}\end{array}}{\overline{f(u_1,\dots,u_k,w)} \in \overline{C(u_1,\dots,u_k,w)}}$$

which is the *+-elimination rule*, and

$$\frac{\overline{u_1} \in \overline{A_1} \quad \ldots \quad \overline{u_k} \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{u} \in \overline{A(u_1,\ldots,u_k)}}{\overline{C(u_1,\ldots,u_k,i(u))} \ni \overline{f(u_1,\ldots,u_k,i(u))} = \overline{a(u_1,\ldots,u_k,u)} \in \overline{C(u_1,\ldots,u_k,i(u))}}$$

$$\frac{\overline{u_1} \in \overline{A_1} \quad \ldots \quad \overline{u_k} \in \overline{A_k(u_1,\ldots,u_{k-1})} \quad \overline{v} \in \overline{B(u_1,\ldots,u_k)}}{\overline{C(u_1,\ldots,u_k,j(v))} \ni \overline{f(u_1,\ldots,u_k,j(v))} = \overline{b(u_1,\ldots,u_k,v)} \in \overline{C(u_1,\ldots,u_k,j(v))}}$$

The forms of type and object expression involved here are translated as follows. We may translate $A + B$ by

- (A, B) in Cantor's notation
- $A + B$
- the disjoint union (sum, coproduct) of the two types A and B
- $A \vee B$
- A or B

we may translate $i(x)$ and $j(y)$ by

- the canonical injection of the object x (resp. y) of the type which is the value of A (resp. B) into $A + B$
- the proof of $A \vee B$ which is obtained from the proof x (resp. y) of the proposition which is the value of A (resp. B)

If

$$\begin{array}{c} \overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{A(u_1, \ldots, u_k)} \text{ is a type} \end{array}$$

then

$$\frac{\begin{array}{l} x_1 \in X_1 = \overline{A}_1 \\ \quad\vdots \\ x_k \in X_k = \overline{A_k(u_1, \ldots, u_{k-1})} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_{k-1} = x_{k-1} \\ x \in X = \overline{A(u_1, \ldots, u_k)} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_k = x_k \end{array}}{r(x) \in I_{A(u_1, \ldots, u_k)}(x, x)}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_k)}}{\overline{r(u)\, I_{A(u_1, \ldots, u_k)}(u, u)}}$$

which is the *I-introduction rule*, and

$$\frac{\begin{array}{l} x_1 \in X_1 = \overline{A}_1 \\ \quad\vdots \\ x_k \in X_k = \overline{A_k(u_1, \ldots, u_{k-1})} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_{k-1} = x_{k-1} \\ x \in X = \overline{A(u_1, \ldots, u_k)} \text{ for } \overline{u}_1 = x_1, \ldots, \overline{u}_k = x_k \\ y \in X = \overline{A(u_1, \ldots, u_k)} \text{ for } \overline{u}_1 = x_k, \ldots, \overline{u} = x \end{array}}{I_{A(x_1, \ldots, x_k)}(x, y) \text{ is a type}}$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \underline{u}_k \in \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \begin{array}{r} \overline{u} \in \overline{A(u_1, \ldots, u_k)} \\ \overline{v} \in \overline{A(u_1, \ldots, u_k)} \end{array}}{\overline{I_{A(u_1, \ldots, u_k)}(u, v)} \text{ is a type}}$$

Furthermore, if

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_k)}}{\overline{c(u_1, \ldots, u_k, u)} \in \overline{C(u_1, \ldots, u_k, u, u, r(u))}}$$

$$\frac{\begin{array}{ccc} \overline{u}_1 \in \overline{A}_1 & \ldots & \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \overline{u} \in \overline{A(u_1, \ldots, u_k)} & \overline{v} \in \overline{A(u_1, \ldots, u_k)} & \overline{w} \in \overline{I_{A(u_1, \ldots, u_k)}(u, v)} \end{array}}{\overline{C(u_1, \ldots, u_k, u, v, w)} \text{ is a type}}$$

and $f(u_1, \ldots, u_k, u, v, w)$ is defined from $c(u_1, \ldots, u_k, u)$ by the *I-scheme*, then

$$\frac{\begin{array}{ccc} \overline{u}_1 \in \overline{A}_1 & \ldots & \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \overline{u} \in \overline{A(u_1, \ldots, u_k)} & \overline{v} \in \overline{A(u_1, \ldots, u_k)} & \overline{w} \in \overline{I_{A(u_1, \ldots, u_k)}(u, v)} \end{array}}{\overline{f(u_1, \ldots, u_k, u, v, w)} \in \overline{C(u_1, \ldots, u_k, u, v(w))}}$$

which is the rule of *I - elimination*, and

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{A(u_1, \ldots, u_k)}}{\overline{C(u_1, \ldots, u_k, u, u, r(u))} \ni \overline{f(u_1, \ldots, u_k, u, u, r(r))} = \overline{c(u_1, \ldots, u_k, u)} \in \overline{C(u_1, \ldots, u_k, u, u, r(u))}}$$

We may translate $I_A(x, y)$ by

- $x =_A y$,
- $x = y$
- the proposition that x and y are identical objects of the type which is the value of A

and $r(x)$ is translated by

- the proof that th object x of the type which is the value of A is identical to itself

From the elimination rule for I we may obtain the usual eliminatory rule for identity

$$\frac{u = v \qquad A(u)}{A(v)}$$

as follows. The latter rule is interderivable with the rule

$$\frac{u = v \qquad C(u, u)}{C(u, v)}$$

This is obvious in one direction, and in the other we may take $C(u, v)$ to be $A(u) \rightarrow A(v)$. The second rule above is a special case of the rule of I-elimination.

6.6 *The N_n-Rule Family*

$$1 \in N_n \qquad \dots \qquad n \in N_n \qquad N_n \text{ is a type}$$

$$\overline{1} \in \overline{N}_n \qquad \dots \qquad \overline{n} \in \overline{N}_n \qquad \overline{N}_n \text{ is a type}$$

The latter are the N_n-***introduction rules***. Furthermore, if

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1, \dots, u_{k-1})}}{\overline{a_m(u_1, \dots, u_k)} \in \overline{A(u_1, \dots, u_k, m)}} \qquad m = 1, \dots, n$$

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1, \dots, u_{k-1})} \quad \overline{u} \in \overline{N}_n}{\overline{A(u_1, \dots, u_k, u)} \text{ is a type}}$$

and $f(u_1, \ldots, u_k, u)$ has been defined from $a_1(u_1, \ldots, u_k), \ldots, a_n(u_1, \ldots, u_k)$ by the N_n-scheme, then

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \ldots \quad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})} \quad \overline{u} \in \overline{N}_n}{\overline{f(u_1, \ldots, u_k, u)} \in \overline{A(u_1, \ldots, u_k, u)}}$$

which is the rule of N_n-***elimination***, and

$$\frac{\overline{u}_1 \in \overline{A}_1 \qquad \ldots \qquad \overline{u}_k \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{A(u_1, \ldots, u_k, m)} \ni \overline{f(u_1, \ldots, u_k, m)} = \overline{a_m(u_1, \ldots, u_k)} \in \overline{A(u_1, \ldots, u_k, m)}}$$

for $m = 1, \ldots, n$.

One may translate N_0 by

- ϕ
- the empty type
- $\perp$
- $\wedge$
- absurdity
- falsehood

One may translate N_1 by

- $\{1\}$
- the one element type

- T

- truth

One may translate N_n by

- $\{1, \ldots, n\}$,

- the standard type with n elements

In case $n = 0$, the elimination rule for N_n is the formal rendering of the informal rule

$$\frac{\perp}{C}$$

called *ex falso quodlibet*, or absurdity elimination.

7. General Rules Of Function Formation

From the rules for introducing simple functions, we build up composite functions by means of the rules

$$\frac{\overline{a_1} \in \overline{A_1} \quad \ldots \quad \overline{a_k} \in \overline{A_k(a_1, \ldots, a_{k-1})}}{\overline{a(a_1 \ldots, a_k)} \in \overline{A(a_1, \ldots, a_k)}}$$

in which it is supposed that

$$\begin{array}{c} \overline{u_1} \in \overline{A_1} \qquad \ldots \qquad \overline{u_k} \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \overline{a(u_1, \ldots, u_k)} \in \overline{A(u_1, \ldots, u_k)} \end{array}$$

and

$$\frac{\overline{a_1} \in \overline{A_1} \qquad \ldots \qquad \overline{a_k} \in \overline{A_k(a_1, \ldots, a_{k-1})}}{\overline{A(a_1 \ldots, a_k)} \text{ is a type}}$$

in which it is supposed that

$$\frac{\overline{u}_1 \in \overline{A}_1 \quad \dots \quad \overline{u}_k \in \overline{A_k(u_1, \dots, u_{k-1})}}{\overline{A(u_1, \dots, u_k)} \text{ is a type}}$$

Next there is the rule of assumptions, according to which if

$\overline{A}$ is a type

then

$\overline{u} \in \overline{A}$

that is, if we have derived $\overline{A}$ is a type, then we make the assumption $\overline{u} \in \overline{A}$. This corresponds in informal mathematics to saying

Let u denote an arbitrary object of the type denoted by A

or simply

Assume A

Finally there is the rule of replacing the range of a function by one to which it is definitionally equal

$$\frac{\overline{a} \in \overline{A} \qquad \overline{A} = \overline{B}}{\overline{a} \in \overline{B}}$$

By this rule, the rules of function formation are linked with the rules definitional equality. We have already given those rules of definitional equality that relate directly to the particular simple functional expressions that we will use in the language to begin with. Besides these are the following.

8. General Rules Of Definitional Equality

$$\frac{\overline{a} \in \overline{A} \qquad \overline{A} = \overline{B}}{\overline{A} \ni \overline{a} = \overline{a} \in \overline{B}} \qquad \text{– Reflexivity}$$

$$\frac{\overline{A} \text{ is a type}}{\overline{A} = \overline{A}}$$

$$\frac{\overline{A} \in \overline{a} = \overline{b} \in \overline{B}}{\overline{B} \ni \overline{b} = \overline{a} \in \overline{A}} \qquad \frac{\overline{A} = \overline{B}}{\overline{B} = \overline{A}} \qquad \text{– Symmetry}$$

$$\frac{\overline{A} \ni \overline{a} = \overline{b} \in \overline{B} \quad \overline{B} \ni \overline{b} = \overline{c} \in \overline{C}}{\overline{A} \ni \overline{a} = \overline{c} \in \overline{C}}$$

$$\frac{\overline{A} = \overline{B} \quad \overline{B} = \overline{C}}{\overline{A} = \overline{C}} \qquad \text{– Transitivity}$$

Preservation under composition

$$\frac{\begin{array}{c} \overline{A_1} \ni \overline{a_1} = \overline{b_1} \in \overline{A_1} \\ \vdots \\ \overline{A_k(a_1, \ldots, a_{k-1})} \ni \overline{a_k} = \overline{b_k} \in \overline{A_k(b_1, \ldots, b_{k-1})} \end{array}}{\overline{A(a_1, \ldots, a_k)} \ni \overline{a(a_1 \ldots, a_k)} = \overline{a(b_1, \ldots, b_k)} \in \overline{A(b_1, \ldots, b_k)}}$$

in which it is supposed that

$$\begin{array}{c} \overline{u_1} \in \overline{A_1} \qquad \ldots \qquad \overline{u_k} \in \overline{A_k(u_1, \ldots, u_{k-1})} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a(u_1, \ldots, u_k) \in A(u_1, \ldots, u_k) \end{array}$$

and

$$\frac{\begin{array}{c} \overline{A_1} \ni \overline{a_1} = \overline{b_1} \in \overline{A_1} \\ \vdots \\ \overline{A_k(a_1, \ldots, a_{k-1})} \ni \overline{a_k} = \overline{b_k} \in \overline{A_k(b_1, \ldots, b_{k-1})} \end{array}}{\overline{A(a_1, \ldots, a_k)} = \overline{A(b_1, \ldots, b_k)}}$$

in which it is supposed that

$$\frac{\overline{u_1} \in \overline{A_1} \quad \ldots \quad \overline{u_k} \in \overline{A_k(u_1, \ldots, u_{k-1})}}{\overline{A(u_1, \ldots, u_k)} \text{ is a type}}$$

9. Conclusion

With this the depth grammar of mathematics is complete. We have given substance to Wittgenstein's claim that "grammar is a 'theory of logical types'",[24] by showing how a philosophical investigation in an extension of Russell's doctrine of types. This is not surprising when one recalls the influence of Russell on Wittgenstein. Furthermore, we have shown how Martin-Löf's intuitionistic theory of types, being an extension of Russell's doctrine of types, can be further extended to become a logical depth grammar in Wittgenstein's sense, in accordance with the ***proposition-as-rules idea.***[25] It meets Wittgenstein's requirement, as expressed by Hidé Ishiguro, that a "theory of types (is) a necessary truth about symbolism and language: something which can be grasped as evident, if we correctly understand the nature of symbolism. It (is) not a philosophical position which we can argue for or against in a non-circular manner. Nor on the other hand is it something we are free to decide by stipulation".[26] These requirements become obvious once it is realized that a philosophical investigation provides the logical depth grammar of mathematical Language which we ***use*** in correct mathematical practice. It provides the ***logical form*** of mathematical Language.

10. Appendix: The Axiom Of Choice

As an example we shall prove the ***axiom of choice*** following Martin - Löf (1973).[27] This is to engage in judging (asserting) a proposition of the form "p is true". We compute the axiom of choice in virtue of a person program which is the depth grammar of mathematical language. In order to prove the axiom of choice

$$(\Pi u \in A)(\Sigma v \in B(u))C(u,v)$$
$$\rightarrow (\Sigma w \in (\Pi u \in A)B(u))(\Pi u \in A)C(u,ap(u,w))$$

we must construct an object of this type, provided

$$\overline{A}\text{ is a type} \qquad \frac{\overline{u} \in \overline{A}}{\overline{(u,v)}\text{ is a type}} \qquad \frac{\overline{u} \in \overline{A} \quad \overline{v} \in \overline{B(u)}}{\overline{C(u,v)}\text{ is a type}}$$

First we define functional expressions $p(w)$ and $q(w)$ by the Σ-scheme so that

$$p((x,y)) = x$$
$$q((x,y)) = y$$

Now assume

$$\overline{u}_1 \in \overline{A}_1 \tag{1}$$

$$\overline{f} \in \overline{(\Pi u \in A)(\Sigma v \in B(u))C(u,v)} \tag{2}$$

Then by one of the Π-rules (rule of application) we get

$$\overline{ap(u,f)} \in \overline{(\Sigma v \in B(u))C(u,v)} \tag{3}$$

Then from this we get by the Σ-rules

$$\overline{p(ap(u,f))} \in \overline{B(u)} \tag{4}$$

$$\overline{q(ap(u,f))} \in \overline{C(u,p(ap(u,f)))} \tag{5}$$

From (4) we get by a Π-rule

$$\overline{(\lambda u)p(ap(u,f))} \in \overline{(\Pi u \in A)B(U)} \tag{6}$$

and by another

$$\overline{B(u)} \ni \overline{ap(u,(\lambda u)p(ap(u,f)))} = \overline{p(ap(u,f))} \in \overline{B(u)} \tag{7}$$

Then by the rule of the preservation of definitional equality under composition

$$\overline{C(u,ap(u,(\lambda u)p(ap)(u,f))))} = \overline{C(u,p(ap(u,f)))} \tag{8}$$

From (5) and (8) we get by the rules about the symmetry of definitional equality and the replacement of a range by one to which it is definitionally equal

$$\overline{q(ap(u,f))} \in \overline{C(u,ap(u,(\lambda u)p(ap(u,f))))} \tag{9}$$

From (9) and a Π-rule

$$\overline{(\lambda u)q(ap(u,f))} \in \overline{(\Pi u \in A)C(u,ap(u,(\lambda u)p(ap(u,f))))} \tag{10}$$

Then from (6) and (10) by a Σ-rule

$$\begin{aligned} &\overline{((\lambda u)p(ap(u,f)),(\lambda u)q(ap(u,f)))} \\ &\in \overline{(\Sigma w \in (\Pi u \in A)B(u))C(u,ap(u,w))} \end{aligned} \tag{11}$$

Finally from (11) by abstraction

$$\begin{aligned} &(\lambda f)((\lambda u)p(ap(u,f)),(\lambda u)q(ap(u,f))) \in (\Pi u \in A) \\ &(\Sigma v \in B(u)) \\ &C(u,v) \rightarrow (\Sigma w \in (\Pi u \in A)B(u))(\Pi u \in A)C(u,ap(u,w)) \end{aligned}$$

Q. E. D.

REFERENCES

1. Martin-Löf, Per., *Syntax and Semantics of Mathematical Language*, notes written by Peter Hancock on half of a lecture series given by Per Martin-Löf at Oxford University in Michaelmas Term 1975 (unpublished).

2. Martin-Löf, Per., *An intuitionistic theory of types: predicative part*, in H. E. Rose and J. C. Sheperdson (eds.), *Logic Colloquium '73*, Amsterdam (1975) *73-118*; Martin-Löf, Per., *Constructive mathematics and computer programming*, in Cohen L. J., Las, J., Pfeiffer, H., and Podewski, K. P. (eds.), *Logic, Methodology, and Philosophy of Science VI* (Amsterdam) forthcoming; The only other attempt to construct a system which differentiates (in principle) between judgements (assertions) as acts of knowledge and judgements (assertions) as objects of knowledge is made by Dana Scott in *Constructive Validity*, in *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics 125, (1970) *237-275*.

3. Russell, Bertrand., *The Principles of Mathematics*, London (1937) *523*.

4. Sundholm, Göran., *Constructions, Proofs and The Meaning of Logical Constants*, Journal of Philosophical Logic *12* (1983) *167*.

5. Kleene, S. C., *On the Interpretation Of Intuitionistic Number Theory*, Journal of Symbolic Logic *10* (1945) *109-124*.

6. Läuchli, H., *An Abstract Notion of Realizability for which Intuitionistic Predicate Calculus is Complete*, in Myhill, Kino and Vesley (eds.), *Intuitionism and Proof Theory*, Amsterdam (1970) *227-234*.

7. Gödel, Kurt., *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes*, Dialectica *12* (1958) *280-287*.

8. Howard, W. A., *The Formulae-As-Types Notion of Construction*, in Seldin and Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, London (1980) *47-9-490.*

9. Gentzen, G., *Untersuchungen über das logische Schliessen*, Mathematische Zeitschrift *39* (1934) *176-210, 405-431.*

10. Prawitz, Dag., *Ideas and results in proof theory*, in Proceedings of the Second Scandinavian Logic Symposium, G. E. Fenstad (ed.), Amsterdam (1971) *235-307.*

11. de Bruijn, N. G., *A Survey of the Project AUTOMATH*, in Seldin and Hindley (eds.), *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, London (1980) *579-606.*

12. Martin-Löf, Per., *Constructive mathematics and computer programming* (cf. n. 2).

13. Gefwert, Christoffer, *The Proposition-As-Rules Idea*, SLAC-PUB 3303 (March 1984), Stanford.

14. Gefwert, Christoffer, *A Participator: The Metaphysical Subject*, SLAC-PUB 3277 (December 1983), Stanford.

15. Maury, André, *Wittgenstein and the Limits of Language*, Acta Philosophica Fennica, Vol. *32*, Helsinki (1981) *151.*

16. Wittgenstein, Ludwig, *Philosophical Grammar*, Oxford (1974) *186.*

17. Gefwert, Christoffer, A Participator: ... (cf. n. 14), *32-35.*

18. Wittgenstein, Ludwig, *Philosophical Remarks*, Oxford (1975) *54.*

19. Martin-Löf, Per., *An intuitionistic theory of types* ... (cf. n. 2), *76*; Bishop, Erret., *Foundations of Constructive Analysis*, New York (1967) *2.*

20. Curry, H. B. and Feys, R., *Combinary Logic*, Vol. I, Amsterdam (1968).

21. Howard, W. A., *Formulae-As-Types Notion of Construction* (cf. n. 8).

22. Martin-Löf, Per., *About Models for Intuitionistic Type-theories and the Nation of Definitional Equality*, in Kanger, S., ed., Proc. 3rd Scand. Logic Symp., Amsterdam (1975) *101*; Gefwert, Christoffer, *On the Logical Form of Primitive Recursive Functions*, SLAC-PUB-3334 (May 1984).

23. Gentzen, G., *Untersuchungen* über *das logische Schliessen*, (cf. n. 9).

24. Wittgenstein, L., *Philosophical Remarks*, Oxford (1975) *54*.

25. Gefwert, Christoffer, *The Proposition-As-Rules Idea*, SLAC-PUB-3303 (March 1984).

26. Ishiguro, Hidé, *Wittgenstein and the Theory of Types, in Block*, J. (ed.), *Perspectives on the Philosophy of Wittgenstein*, Oxford (1981), *47*.

27. Martin-Löf, Per., *An intuitionistic theory of types* ... (cf. n. 2), *97-98*.