AN INTUITIONISTIC THEORY OF TYPES: PREDICATIVE PART

Per MARTIN-LÖF

University of Stockholm, Stockholm, Sweden

The theory of types with which we shall be concerned is intended to be a full scale system for formalizing intuitionistic mathematics as developed, for example, in the book by Bishop[1]. The language of the theory is richer than the languages of traditional intuitionistic systems in permitting proofs to appear as parts of propositions so that the propositions of the theory can express properties of proofs (and not only individuals, like in first order predicate logic). This makes it possible to strengthen the axioms for existence, disjunction, absurdity and identity. In the case of existence, this possibility seems first to have been indicated by Howard[10], whose proposed axioms are special cases of the existential elimination rule of the present theory. Furthermore, there are axioms for universes (in the sense of category theory) which link the generation of objects and types and play somewhat the same role for the present theory as does the replacement axiom for Zermelo–Fraenkel set theory. They also make the theory adequate for the formalization of certain constructions in category theory, like the construction of the category of all small categories.

An earlier, not yet conclusive, attempt at formulating a theory of this kind was made by Scott[19]. Also related, although less closely, are the type and logic free theories of constructions of Kreisel[12, 13] and Goodman[8].

In its first version, the present theory was based on the strongly impredicative axiom that there is a type of all types whatsoever, in symbols, $V \in V$, which is at the same time a type and an object of that type. This axiom had to be abandoned, however, after it had been shown

to lead to a contraction by Girard [7]. I am very grateful to Jean-Yves Girard for the discovery of the paradox.

The incoherence of the idea of a type of all types whatsoever made it necessary to distinguish, like in category theory, between small and large types. Thus the universe V appears, not as the type of all types, but as the type of small types, whereas V itself and all types which are built up from it are large. This makes the types wellfounded and the theory predicative. However, even in this second form, a serious defect remained: the definition of convertibility, the formal counterpart of definitional equality, turned out to be too liberal in allowing unrestricted conversions under the lambda sign and other variable binding operators. The reasons for this and the corrections that have to be made are all explained in Martin-Löf [16].

In this third version, identity has been added as a primitive, the universe V has been extended to a whole sequence

$$V =_{\text{def}} V_0 \in V_1 \in \cdots \in V_n \in \cdots,$$

which gives the theory more natural closure properties, and the definition of convertibility has been corrected. This has only been possible by abandoning bound variables altogether, except in certain informal metanotations. As a result of the changes in the definitions of convertibility and reduction, the previous defects (pointed out in Martin-Löf [16]) in the proof of normalization are removed. Also, because of the type restrictions on the rules of conversion and reduction, the method for proving the Church–Rosser property developed in combinatory logic apparently no longer works. Instead, the uniqueness of normal form and the Church–Rosser property are proved, almost without effort, as corollaries to the construction of the term model by a new method, due to Peter Hancock.

As for the proof theoretic strength of the theory, it has been conjectured by Peter Hancock that the ordinal of the theory based on the first n universes equals, in Veblen's notation,

$$\phi\,\phi\,n\,\phi_{\epsilon_0}(0)\;(0)\;(0)$$

In particular, for $n = 0$, that is, for the theory without universes, we get just ϵ_0, whereas, for the full theory, we get

$$\lim_n \phi\,\phi\,n\,\phi_{\epsilon_0}(0)\;(0)\;(0) = \Gamma_0,$$

which is the ordinal of predicative analysis determined by Schütte [20, 21] and Feferman [5].

1. Informal explanations of the primitive notions

1.1. *Mathematical objects and their types*

We shall think of *mathematical objects* or *constructions*. Every mathematical object is of a certain kind or *type*. Better, a mathematical object is always given together with its type, that is, it is not just an object: it is an object of a certain type. This may be regarded as a simpler and at the same time more general formulation of Russell's *doctrine of types* [18], according to which a type is the range of significance of a propositional function, because, in the theory that I am about to describe, 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 [1]. Put differently, a type is welldefined if we understand (or *grasp* to use a word favoured by Kreisel [15]) what it means to be an object of that type. Thus, for instance, $N \to N$ is a type, not because we know particular number theoretic functions like the primitive recursive ones, but because we think we understand the notion of number theoretic function *in general*. Note that it is required, neither that we should be able to generate somehow all objects of a given type, nor that we should so to say know them all individually. It is only a question of understanding what it means to be an *arbitrary* object of the type in question. I shall use the notation

$$a \in A$$

to express that

$$a \text{ is an object of type } A.$$

1.2. *Propositions and proofs*

A *proposition* is defined by prescribing how we are allowed to prove it, and a proposition *holds* or is *true* intuitionistically if there is a proof of it. For example,

$$971 \text{ is a nonprime number}$$

is the proposition which we prove by exhibiting two natural numbers greater than one and a computation which shows that their product equals 971. In the present context, however, it will not be necessary to introduce the notion of proposition as a separate notion because we can represent

each proposition by a certain type, namely, the type of proofs of that proposition. That the proofs of a proposition must form a type is inherent already in the intuitionistic idea of considering proofs as mathematical objects when taken together with the doctrine of types. Indeed, if the proofs of a proposition are to be considered as mathematical objects, they must form a type, namely, the type of proofs of the proposition in question.

Conversely, each type determines a proposition, namely, the proposition that the type in question is nonempty. This is the proposition which we prove by exhibiting an object of the type in question. On this analysis, there appears to be no fundamental difference between propositions and types. Rather, the difference is one of point of view: in the case of a proposition, we are not so much interested in what its proofs are as in whether it has a proof, that is, whether it is true or false, whereas, in the case of a type, we are of course interested in what its objects are and not only in whether it is empty or nonempty.

When a type A is thought of as a proposition,

$$a \in A$$

expresses that

$$a \text{ is a proof of the proposition } A.$$

On the formal level, the analogy between formulae and type symbols was discovered by Curry and Feys [3] and further extended by Howard [10] to whom I am indebted for explaining it to me. In what follows, I shall make use of it in much the same way as Scott [19].

1.3. *Properties*

A propositional function defined on a type A is called a *property* or, in intuitionistic terminology, a *species* of objects of type A. Because of the correspondence between propositions and types, it will sometimes be convenient to call a type valued function defined on A a species of objects of type A as well. Note that the notion of property is not a primitive notion: it is explained in terms of the notion of proposition (or type) and the notion of function. If a is an object of type A and B a species of objects of type A, then $B(a)$ is the proposition that a *belongs* to the species B. This proposition will invariably be written $B(a)$ in order to avoid confusion between the belonging relation and the relation

between an object and its type for which we have reserved the epsilon notation. Thus, for example, if $\mathbf{N}$ denotes the type of natural numbers, then $3 \in \mathbf{N}$ expresses that 3 is a natural number, whereas the proposition that 3 is a prime number is denoted by $P(3)$, assuming of course that P denotes the property of being a prime number (which is a property of natural numbers).

1.4. *Cartesian product of a family of types*

Suppose now that A is a type and that B is a function which to an arbitrary object x of type A assigns a type $B(x)$. Then the *cartesian product*

$$(\Pi x \in A)B(x)$$

is a type, namely, the type of *functions* (*rules* or *methods*) which take an arbitrary object x of type A into an object of type $B(x)$. Clearly, we may *apply* an object b of type $(\Pi x \in A)B(x)$ to an object a of type A, thereby getting an object

$$b(a)$$

of type $B(a)$. When $B(x)$ represents a proposition for every object x of type A, $(\Pi x \in A)B(x)$ represents the *universal* proposition

$$(\forall x \in A)B(x).$$

A proof of $(\forall x \in A)B(x)$ is a function which to an arbitrary object x of type A assigns a proof of $B(x)$.

Functions may be introduced by *explicit definition*. That is, if we, starting from a variable x which denotes an arbitrary object of type A, build up an expression $b[x]$ which denotes an object of type $B(x)$, then we may define a function f of type $(\Pi x \in A)B(x)$ by the schema

$$f(x) =_{\text{def}} b[x].$$

The square brackets are used to indicate the occurrences of the variable x in the expression $b[x]$.

If $B(x)$ is defined to be one and the same type B for every object x of type A, then $(\Pi x \in A)B(x)$ will be abbreviated

$$A \to B.$$

It is the type of functions from A to B. When A and B both represent propositions, $A \to B$ represents the *implication*

$$A \supset B.$$

A proof of $A \supset B$ is a function which takes an arbitrary proof of A into a proof of B.

1.5. *Disjoint union of a family of types*

Given a type A and a function B which to an arbitrary object x of type A assigns a type $B(x)$, we may form the *disjoint union*

$$(\Sigma x \in A)B(x)$$

which is the type of pairs (x, y) where x and y are objects of types A and $B(x)$, respectively. When $B(x)$ represents a proposition for every object x of type A, $(\Sigma x \in A)B(x)$ represents the *existential* proposition

$$(\exists x \in A)B(x)$$

which we prove by exhibiting a pair (x, y) consisting of an object x of type A and a proof y of the proposition $B(x)$.

Let C be a function which to an arbitrary object of type $(\Sigma x \in A)B(x)$ assigns a type. Given a binary function g which to a pair of objects x and y of types A and $B(x)$, respectively, assigns an object of type $C((x, y))$, we may then define a unary function f which to an object z of type $(\Sigma x \in A)B(x)$ assigns an object of type $C(z)$ by the schema

$$f((x, y)) =_{\text{def}} g(x, y).$$

In particular, we can define the left and right projections by putting

$$\begin{cases} p((x, y)) =_{\text{def}} x, \\ q((x, y)) =_{\text{def}} y. \end{cases}$$

Clearly, p and q take an object z of type $(\Sigma x \in A)B(x)$ into objects of types A and $B(p(z))$, respectively.

A third function of $(\Sigma x \in A)B(x)$ is to represent the type of all objects x of type A *such that* $B(x)$, because, from the intuitionistic point of view, to give an object x of type A such that $B(x)$ is to give x *together* with a proof y of the proposition $B(x)$. This interpretation of the notion of such that is implicitly used by Bishop[1] and discussed by Kreisel[14]. However, its explicit formulation compels us to regard proofs as

mathematical objects. For example, the type R of real numbers is defined as

$$(\Sigma\, x \in N \to Q)(\Pi\, m \in N)(\Pi\, n \in N)(|x_{m+n} - x_m| \leq 2^{-m}).$$

Thus a real number is a pair (x, y), where x is a sequence of rational numbers and y is a proof that x satisfies the Cauchy condition.

An example which shows the necessity of treating proofs as mathematical objects is afforded by the inverse function which is not of type $R \to R$ but of type $(\Pi\, z \in R)\,(z \neq 0 \to R)$, because the definition of the inverse z^{-1} of a nonzero real number z depends effectively on the proof that $z \neq 0$. A similar phenomenon occurs in the intuitionistic theory of ordinals of the second number class where the subtraction function is not of type $O \to (O \to O)$ but of type $(\Pi\, x \in O)(\Pi\, y \in O)(x < y \to O)$, because the definition of the difference $y - x$ of two ordinals x and y depends effectively on the proof that $x < y$.

In the special case when $B(x)$ is defined to be one and the same type B for every object x of type A, $(\Sigma\, x \in A)B(x)$ is abbreviated

$$A \times B.$$

It is the *cartesian product* of the two types A and B. If A and B both represent propositions, then $A \times B$ represents their *conjunction*

$$A \;\&\; B,$$

a proof of which is a pair consisting of a proof of A and a proof of B.

1.6. *Disjoint union of two types*

If A and B are types, so is the *disjoint union*

$$A + B$$

which is the type of objects of the form $i(x)$ with x of type A or $j(y)$ with y of type B. Here i and j denote the canonical injections. When A and B both represent propositions, $A + B$ represents their *disjunction*

$$A \vee B,$$

a proof of which consists of either a proof of A (together with the information that it is A that has been proved) or a proof of B (together with the information that it is B that has been proved).

Let C be a function which to an object of type $A + B$ assigns a type,

and suppose that g is a function which to an object x of type A assigns an object of type $C(i(x))$ and that h is a function which to an object y of type B assigns an object of type $C(j(y))$. Then we may define a function f which to an object z of type $A + B$ assigns an object of type $C(z)$ by the schema

$$\begin{cases} f(i(x)) =_{\text{def}} g(x), \\ f(j(y)) =_{\text{def}} h(y). \end{cases}$$

1.7. *Identity*

If x and y are objects of one and the same type A, then

$$I(x, y)$$

is a proposition, namely, the proposition that x and y are *identical*. This is the proposition which is defined by stipulating that, if x is an arbitrary object of type A, then $r(x)$ is a proof of $I(x, x)$. Thus r is our notation for the introductory axiom of identity $(\forall x \in A)I(x, x)$.

Let C be a ternary function which to an arbitrary pair of objects x and y of type A and a proof of $I(x, y)$ assigns a type. Given a unary function g which to an object x of type A assigns an object of type $C(x, x, r(x))$, we may then define a ternary function f which to a pair of objects x and y of type A and a proof z of $I(x, y)$ assigns an object of type $C(x, y, z)$ by the schema

$$f(x, x, r(x)) =_{\text{def}} g(x).$$

In particular, if $C(x)$ represents a proposition for x of type A and g is the function which to an object x of type A assigns the obvious proof of $C(x) \supset C(x)$, that is, the identity function on the type $C(x)$, then the function f introduced by the schema above is the intended proof of the usual eliminatory axiom of identity

$$(\forall x \in A)(\forall y \in A)(I(x, y) \supset (C(x) \supset C(y))).$$

1.8. *Finite types*

For each nonnegative integer n, we introduce a type N_n with precisely the n objects $1, 2, \ldots, n$. Actually, it would suffice to introduce N_0 and N_1, because, for n greater than one, we can define N_n to be the disjoint union of N_1 with itself n times.

If C is a function which to an arbitrary object x of type N_n assigns a

type and $c_1, \ldots, c_n$ are objects of types $C(1), \ldots, C(n)$, respectively, then we may define a function f which to an object x of type N_n assigns an object of type $C(x)$ by the schema

$$\begin{cases} f(1) =_{\text{def}} c_1, \\ \quad \vdots \\ f(n) =_{\text{def}} c_n. \end{cases}$$

In particular, for $n = 0$, N_0 is the *empty type* $\emptyset$ which also represents the logical constant *falsehood* $\bot$, and the function f defined by the above schema (which in this case is vacuous) is the empty function. Similarly, the one element type N_1 is used to represent the logical constant *truth* T.

Negation is defined as usual by

$$-A =_{\text{def}} A \to \bot$$

and is hence not taken as a primitive notion. A proposition A is *false* if $-A$ is true, that is, if there is a proof of $A \to \bot$. Note that this is an affirmative statement.

1.9. *Natural numbers*

N is a type, namely, the type of *natural numbers*. 0 is an object of type N, and, if x is an object of type N, so is its successor $s(x)$. These are the first two Peano axioms.

Let C be a function which to an arbitrary natural number assigns a type. Then, given an object c of type $C(0)$ and a binary function g which to a natural number x and an object of type $C(x)$ assigns an object of type $C(s(x))$, we may define a unary function f which to a natural number x assigns an object of type $C(x)$ by the *recursion* schema

$$\begin{cases} f(0) =_{\text{def}} c, \\ f(s(x)) =_{\text{def}} g(x, f(x)). \end{cases}$$

If $C(x)$ represents a proposition for every natural number x, then f is the proof of the universal proposition $(\forall x \in \mathbf{N})C(x)$ which we get by applying the principle of mathematical *induction* to the proof c of $C(0)$ and the proof g of $(\forall x \in \mathbf{N})(C(x) \supset C(s(x)))$.

1.10. *Universes*

The abstractions described so far still do not allow us to define enough types and type valued functions. For example, we want to be able to

define the type of finite sequences of natural numbers as $(\Sigma x \in N)F(x)$, where

$$\begin{cases} F(0) =_{\text{def}} N_1, \\ F(s(x)) =_{\text{def}} F(x) \times N. \end{cases}$$

This can clearly be done by recursion if only the types N_1 and $\mathbf{N}$ were objects of some type V with the closure property that, if A and B are objects of type V, so is $A \times B$. Also, we want to be able to define transfinite types like $(\Pi x \in \mathbf{N})G(x)$, where

$$\begin{cases} G(0) =_{\text{def}} N, \\ G(s(x)) =_{\text{def}} G(x) \to N. \end{cases}$$

Again, this offers no difficulty if only there were a type V such that N is an object of type V and $A \to B$ is an object of type V as soon as A and B are objects of type V.

Guided by these heuristic considerations, we introduce a type V which will be called a *universe* and whose objects are to be types together with the *reflection principle* which roughly speaking says that whatever we are used to doing with types can be done inside the universe V. More precisely, this means that V is closed with respect to the following inductive clauses. If A is an object of type V and B is a function which to an arbitrary object of type A assigns an object of type V, then $(\Pi x \in A)B(x)$ and $(\Sigma x \in A)B(x)$ are objects of type V. If A and B are objects of type V, so is $A + B$. If A is an object of type V and a and b are objects of type A, then $I(a, b)$ is an object of type V. $N_0, N_1, \ldots$ and N are objects of type V. Note, however, that the reflection principle does *not* justify the axiom that V is an object of type V which Girard [7] has shown to be contradictory, because then V would so to say have to have been there already before we introduced it.

It is not natural although possible to add the principle of transfinite induction on V, expressing the idea that V is the least type which is closed with respect to the above inductive clauses, because we want to keep our universe V open so as to be able to throw new types into it or require it to be closed with respect to new type forming operations. For example, we may want it to include the type O of ordinals of the second number class or to be closed with respect to the operation which to a type A assigns the type $W(A)$ of wellfounded trees over A (see Tait [23], Scott [19] and Howard [11]).

Borrowing terminology from category theory, a type which is an object

of type V is said to be *small* whereas V itself and all types which are built up from it are *large*. Thus the universe V is the type of small types. With this distinction between small and large, the present theory, despite its limited proof theoretic strength, is adequate for the formulation of the basic notions and constructions of category theory such as the construction of the category of all small categories. However, it does not legitimatize the construction of the category of all categories whatsoever which, in view of Girard's paradox, seems highly dubious.

The use of the reflection principle in the present theory, on the one hand, to overcome the unnatural limitation to the finite type structure over the basic types and, on the other hand, to make possible the formalization of category theory should be compared to the use of the quite different reflection principle in the equally different language of set theory for the same purposes. The idea of using the set theoretic reflection principle for the formalization of category theory is due to Kreisel[12] and has been elaborated by Feferman[5].

The idea of introducing a universe can be iterated so as to obtain a whole sequence of universes

$$V =_{\text{def}} V_0, V_1, \ldots, V_n, \ldots .$$

As before, $V =_{\text{def}} V_0$ is the type of small types, whereas V_1 is the type of large types, V_2 the type of extra large types, and so on. Generally, an object of type V_n is said to be a type of *order n*. It will also be convenient to say that a function from a type A into the universe V_n is an n^{th} order property (or species) of objects of type A.

The axioms governing the sequence of universes are obtained by extending in the natural way those for small and large types. Thus, for every n, V_n is a type of order $n+1$, in symbols,

$$V_n \in V_{n+1}.$$

If A is a type of order m and B is a function from A into V_n, then $(\Pi x \in A)B(x)$ and $(\Sigma x \in A)B(x)$ are types of order $\max(m, n)$. The order of $A + B$ is the maximum of the orders of A and B. If A is a type of order n, then the identity relation on A is also of order n, that is, $I(a, b)$ is an object of type V_n for arbitrary objects a and b of type A. Finally, the basic types $N_0, N_1, \ldots$ and N are all of order 0.

With the sequence of universes, we achieve that every type is at the same time an object and, as such, has itself got a type. The difference

between having just one universe and a whole countable sequence of them is like the difference in set theory between the simple axiom that there exists a universe and Grothendieck's axiom that every set is contained in a universe.

1.11. *Definitional equality*

The relation of definitional equality, denoted by $=_{\text{def}}$, has already appeared in the definitional schemata associated with each of the basic types and type forming operations. A definitional schema consists of a certain number of defining equations. As usual, call the left hand member of a defining equation the *definiendum* and the right hand member the corresponding *definiens*. Definitional equality is defined to be the equivalence relation, that is, reflexive, symmetric and transitive relation, which is generated by the principles that a definiendum is always definitionally equal to its definiens and that definitional equality is preserved under substitution. The latter means that, if we replace a part of a mathematical expression by a definitionally equal expression, then the resulting expression is definitionally equal to the one we started with. In schematic form, we may write the rules determining the relation of definitional equality as follows:

$$\text{definiendum} =_{\text{def}} \text{definiens}, \qquad \frac{a =_{\text{def}} c}{b[a] =_{\text{def}} b[c]}$$

$$a =_{\text{def}} a, \qquad \frac{a =_{\text{def}} b}{b =_{\text{def}} a}, \qquad \frac{a =_{\text{def}} b \qquad b =_{\text{def}} c}{a =_{\text{def}} c}.$$

The only way in which the notion of definitional equality enters into a mathematical construction or proof, except in the definitional schemata themselves, is in applications (usually unconscious) of the principle

> if a is an object of type A and $A =_{\text{def}} B$, then a is an object of type B,

and, correspondingly for propositions and proofs,

> if a is a proof of the proposition A and $A =_{\text{def}} B$, then a is a proof of the proposition B.

This principle, which is quite indispensable, might be called the *principle of replacing a type (proposition) by a definitionally equal type (proposition)*. As a typical application of it, consider the following construction.

Define by recursion

$$\begin{cases} F(0) =_{\text{def}} N, \\ F(s(x)) =_{\text{def}} F(x) \to F(x). \end{cases}$$

Then, if g is a function of type $(\Pi\, x \in N)F(x)$, we may define a function f of the same type by the explicit definition

$$f(x) =_{\text{def}} g(s(x))(g(x)).$$

Of course, this presupposes that $g(s(x))(g(x))$ denotes an object of type $F(x)$ for an arbitrary natural number x. This, on the other hand, is seen as follows. Since g is a function of type $(\Pi\, x \in \mathbf{N})F(x)$ and $s(x)$ is a natural number,

$$g(s(x)) \text{ is an object of type } F(s(x)).$$

But $F(s(x)) =_{\text{def}} F(x) \to F(x)$ and hence, by the principle of replacing a type by a definitionally equal type,

$$g(s(x)) \text{ is an object of type } F(x) \to F(x).$$

Finally, $g(x)$ being an object of type $F(x)$, we can apply $g(s(x))$ to $g(x)$, thereby getting an object $g(s(x))(g(x))$ of type $F(x)$.

Another application of the principle is in the proof of the fact (to be used in the proof of uniqueness of normal form) that, for two objects a and b of some type A,

$$\text{if } a =_{\text{def}} b, \text{ then } I(a, b) \text{ is true, that is, } a \text{ and } b \text{ are identical.}$$

The argument is this. First we conclude from $a =_{\text{def}} b$ that $I(a, a) =_{\text{def}} I(a, b)$. But $I(a, a)$ is an axiom and hence, by the principle of replacing a proposition by a definitionally equal proposition, we can conclude $I(a, b)$ as desired. (More pedantically, from the fact that $r(a)$ is a proof of $I(a, a)$, the principle allows us to conclude that $r(a)$ is a proof of $I(a, b)$ and hence that $I(a, b)$ holds.)

2. Formalization of an intuitionistic theory of types

2.1. *Terms and type symbols*

The formal system that we shall erect consists of a certain number of mechanical rules for deriving symbolic expressions of the forms

$$a \in A$$

and

$$a \text{ conv } b$$

which are to be read a is a *term with type symbol* A and a *converts* into b, respectively. In the intended interpretation, $a \in A$ will mean that a is an object of type A and a conv b that $a =_{\text{def}} b$. A rule is classified as a rule of *term formation* or a rule of *conversion* depending on whether its conclusion is of the form $a \in A$ or a conv b. If, for some A, a is a term with type symbol A, then a is a *term*. The rules of conversion are such that it is immediately clear that, parallel to a derivation of a conv b, there run derivations of $a \in A$ and $b \in B$ for some A and B. Thus the convertibility relation is a relation between terms. Keeping this in mind, it will also be clear that, if a is a term with type symbol A, then A is a term. Thus the epsilon relation is likewise a relation between terms.

If A is a term with type symbol V_n, that is, if

$$A \in V_n$$

has been derived, then A is a *type symbol of order n*. And, if A is a type symbol of order n for some $n = 0, 1, \ldots$, then A is a *type symbol*.

2.2. *Variables*

Given a type symbol A, we may introduce a *variable* x with type symbol A. Variables will preferably be denoted by the letters $x, y, z, \ldots$, possibly with subscripts. The type symbol of a variable must be uniquely associated with the variable in question. Thus x cannot be a variable with type symbols A as well as B unless A and B are (syntactically) identical.

A variable x with type symbol A is a term with type symbol A, in symbols,

$$x \in A.$$

This is the first rule of term formation. Writing the symbol string $x \in A$

corresponds to saying informally

let x be an arbitrary object of type A.

If a derivation is viewed as a tree like arrangement of expressions of the forms $a \in A$ and a conv b, then the topmost expressions of the form $x \in A$ are called the *assumptions* of the derivation in question. This is in analogy with the terminology introduced by Gentzen [6] for his system of natural deduction for first order predicate logic.

Every derivation *depends* on a certain number (possibly zero) of variables. If a derivation of $a \in A$ depends on certain variables, we shall also say that the term a depends on these variables. In particular, a type symbol A is said to depend on the variables on which the derivation of $A \in V_n$, which shows that it is a type symbol, depends. The notion of dependence is defined inductively by stipulating that, if $x \in A$ is an assumption of a derivation, then this derivation depends on the variable x as well as on all the variables on which the type symbol A depends. A derivation is *closed* if it depends on no variables at all or, equivalently, if it contains no assumptions. A term a is closed if the derivation of $a \in A$, which shows that it is a term, is closed. In particular, a type symbol A is closed if the derivation of $A \in V_n$, which shows that it is a type symbol, is closed. The rules of term formation and conversion are such that no variables can occur in $a \in A$ or a conv b except those on which the derivation of $a \in A$ respectively a conv b depends. Hence a closed term or type symbol is variable free.

If $x_1, ..., x_k$ is a sequence of variables which contains all those on which a term a depends, we shall sometimes denote a by $a[x_1, ..., x_k]$ in order to indicate that these variables (and no others) may occur in a. Moreover, we shall always assume in such cases that the variables $x_1, ..., x_k$ have the property that, for $i = 1, ..., k$, the type symbol $A_i[x_1, ..., x_{i-1}]$ of the variable x_i depends on no other variables than the preceding ones $x_1, ..., x_{i-1}$.

The result of substituting $a_1, ..., a_k$ for all occurrences of the variables $x_1, ..., x_k$ in $a[x_1, ..., x_k]$ will be denoted by

$$a[a_1, ..., a_k].$$

Here it is supposed, of course, that $a_1, ..., a_k$ are terms with type symbols $A_1, ..., A_k[a_1, ..., a_{k-1}]$, respectively.

2.3. *Constants*

In addition to the variables, there are *function constants*, preferably denoted by $f, g, h, ...$. Each function constant f has associated with it not only a place index k, indicating the number of argument places, but also type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}] \quad \text{and} \quad A[x_1, ..., x_k],$$

indicating the successive types of its arguments and the type of its value for these arguments, respectively. Note that, because of our notational convention, it is here tacitly assumed that $A_i[x_1, ..., x_{i-1}]$ depends on no other variables than $x_1, ..., x_{i-1}$ for $i = 1, ..., k$ and that $A[x_1, ..., x_k]$ depends on no other variables than $x_1, ..., x_k$. A 0-ary function constant is called simply a *constant*. Note that the type symbol of a constant is necessarily closed. It will also be convenient to call a function constant the type of whose value is indicated by one of the symbols V_n, $n = 0, 1, ...$, a *type valued constant* or, in the case of zero argument places, simply a *type constant*. Type valued constants will preferably be denoted by $F, G, H, ...$.

If f is a k-ary function constant whose arguments and value have type indications as above and $a_1, ..., a_k$ are terms with type symbols $A_1, ..., A_k[a_1, ..., a_{k-1}]$, respectively, then $f(a_1, ..., a_k)$ is a term with type symbol $A[a_1, ..., a_k]$. This is the second rule of term formation which we may write schematically as

$$\frac{a_1 \in A_1 \ ... \ a_k \in A_k[a_1, ..., a_{k-1}]}{f(a_1, ..., a_k) \in A[a_1, ..., a_k]}$$

With each of the basic types (except the universes V_n) and type forming operations, there are associated four rules. The first, called the reflection principle, allows us to introduce a type valued constant as a notation for the type forming operation in question and asserts, in addition, that the universes are closed with respect to this operation. In the case of a basic type, this will be a type constant which will have the appropriate V_n as its type symbol. The second and third rules are the introduction and elimination rules (in the sense of Gentzen [6]) associated with the type or type forming operation in question. The fourth is the rule of conversion which links the introduction and elimination rules and is the formal counterpart of the definitional schema associated with the type or type forming operation that we are dealing with.

2.4. *Rules for* Π

Π-*reflection.* If

$$x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}], \quad x \in A[x_1, \ldots, x_k],$$
$$A[x_1, \ldots, x_k] \in V_m \quad \text{and} \quad B[x_1, \ldots, x_k, x] \in V_n,$$

then we may introduce a function constant F whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad V_{\max(m,n)},$$

respectively, and which is to be uniquely associated with $A[x_1, \ldots, x_k]$, $B[x_1, \ldots, x_k, x]$ and the symbol Π. The meaning of F will be clear from the fact that, for $a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}]$, we are going to use

$$(\Pi x \in A[a_1, \ldots, a_k])B[a_1, \ldots, a_k, x]$$

as an informal metanotation for $F(a_1, \ldots, a_k)$. The use of $(\Pi x \in A[a_1, \ldots, a_k])B[a_1, \ldots, a_k, x]$, in which the variable x is bound, as a formal notation in the system leads to serious difficulties in the definition of conversion which seem to be avoided best by writing it as $F(a_1, \ldots, a_k)$.

Π-*introduction.* If $b[x_1, \ldots, x_k, x] \in B[x_1, \ldots, x_k, x]$ depends on no other variables than $x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}]$ and $x \in A[x_1, \ldots, x_k]$, then we may introduce a k-ary function constant f by the schema of functional *abstraction*

$$f(x_1, \ldots, x_k)(x) \text{ conv } b[x_1, \ldots, x_k, x].$$

The type symbols of the arguments of f are apparent, and its value has type symbol $(\Pi x \in A[x_1, \ldots, x_k])B[x_1, \ldots, x_k, x]$. The variables $x_1, \ldots, x_k$ are called the *parameters* of the schema. For $a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}]$, it will sometimes be convenient to use the expression

$$(\lambda x)b[a_1, \ldots, a_k, x],$$

in which the variable x is bound, as an informal metanotation for $f(a_1, \ldots, a_k)$.

Π-*elimination.* If

$$a \in A[a_1, \ldots, a_k] \quad \text{and} \quad c \in (\Pi x \in A[a_1, \ldots, a_k])B[a_1, \ldots, a_k, x],$$

then $c(a) \in B[a_1, \ldots, a_k, a]$. This is the rule of functional *application.*

Π-*conversion.* If

$$a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}] \quad \text{and} \quad a \in A[a_1, \ldots, a_k]$$

and the function constant f has been introduced by the above schema of abstraction, then

$$f(a_1, \ldots, a_k)(a) \text{ conv } b[a_1, \ldots, a_k, a]$$

or, using the lambda notation,

$$(\lambda x)b[a_1, \ldots, a_k, x](a) \text{ conv } b[a_1, \ldots, a_k, a].$$

2.5. *Rules for* Σ

Σ-*reflection.* If

$$x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad x \in A[x_1, \ldots, x_k]$$

are variables with type symbols as indicated and $A[x_1, \ldots, x_k] \in V_m$ and $B[x_1, \ldots, x_k, x] \in V_n$, then we may introduce a function constant F whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad V_{\max(m,n)},$$

respectively, and which is to be uniquely associated with $A[x_1, \ldots, x_k]$, $B[x_1, \ldots, x_k, x]$ and the symbol Σ. For $a_1 \in A_1$, $a_k \in A_k[a_1, \ldots, a_{k-1}]$, we shall use

$$(\Sigma x \in A[a_1, \ldots, a_k])B[a_1, \ldots, a_k, x]$$

as an informal metanotation for $F(a_1, \ldots, a_k)$.

Σ-*introduction.* Associated with $(\Sigma x \in A[x_1, \ldots, x_k])B[x_1, \ldots, x_k, x]$, there is a $(k+2)$-ary *pairing* constant with arguments and value of types

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}], \quad A[x_1, \ldots, x_k], \quad B[x_1, \ldots, x_k, x]$$
$$\text{and} \quad (\Sigma x \in A[x_1, \ldots, x_k])B[x_1, \ldots, x_k, x],$$

respectively. The result of applying the pairing constant to the terms $a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}]$, $a \in A[a_1, \ldots, a_k]$ and $b \in B[a_1, \ldots, a_k, a]$ will be denoted by

$$(a_1, \ldots, a_k, a, b)$$

which, in the case $k = 0$ of no parameters, reduces to the standard notation (a, b).

Σ-elimination. If

$$x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}],$$
$$x \in A[x_1, \ldots, x_k] \quad \text{and} \quad y \in B[x_1, \ldots, x_k, x]$$

are variables with the indicated type symbols and

$$c[x_1, \ldots, x_k, x, y] \in C[x_1, \ldots, x_k, (x_1, \ldots, x_k, x, y)]$$

where $C[x_1, \ldots, x_k, z]$ is a type symbol which, in addition to the parameters $x_1, \ldots, x_k$ depends on a variable

$$z \in (\Sigma\, x \in A[x_1, \ldots, x_k])B[x_1, \ldots, x_k, x],$$

then we may introduce a $(k+1)$-ary function constant f whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}],$$
$$(\Sigma\, x \in A[x_1, \ldots, x_k])B[x_1, \ldots, x_k, x] \quad \text{and} \quad C[x_1, \ldots, x_k, z],$$

respectively, by the schema

$$f(x_1, \ldots, x_k, (x_1, \ldots, x_k, x, y)) \text{ conv } c[x_1, \ldots, x_k, x, y].$$

Σ-conversion. If f is a function constant which has been introduced by this schema and

$$a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}],$$
$$a \in A[a_1, \ldots, a_k] \quad \text{and} \quad b \in B[a_1, \ldots, a_k, a],$$

then

$$f(a_1, \ldots, a_k, (a_1, \ldots, a_k, a, b)) \text{ conv } c[a_1, \ldots, a_k, a, b].$$

2.6. *Rules for +*

+-reflection. If $x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}]$ are variables with the indicated type symbols and $A[x_1, \ldots, x_k] \in V_m$ and $B[x_1, \ldots, x_k] \in V_n$, then we may introduce a function constant F whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad V_{\max(m,n)},$$

respectively, and which is to be uniquely associated with $A[x_1, \ldots, x_k]$, $B[x_1, \ldots, x_k]$ and the symbol $+$. For $a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}]$, we shall use

$$A[a_1, \ldots, a_k] + B[a_1, \ldots, a_k]$$

as an informal metanotation for $F(a_1, \ldots, a_k)$.

+-introduction. With each type symbol of the form $A[x_1, \ldots, x_k] + B[x_1, \ldots, x_k]$, there are associated $(k+1)$-ary *injection* constants i and j whose arguments and values have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}], \quad A[x_1, \ldots, x_k]$$
$$\text{and} \quad A[x_1, \ldots, x_k] + B[x_1, \ldots, x_k]$$

and

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}], \quad B[x_1, \ldots, x_k]$$
$$\text{and} \quad A[x_1, \ldots, x_k] + B[x_1, \ldots, x_k],$$

respectively.

+-elimination. Suppose that

$$x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}],$$
$$x \in A[x_1, \ldots, x_k] \quad \text{and} \quad y \in B[x_1, \ldots, x_k]$$

are variables with the indicated type symbols and that $C[x_1, \ldots, x_k, z]$ is a type symbol which depends not only on $x_1, \ldots, x_k$ but also (possibly) on a variable $z \in A[x_1, \ldots, x_k] + B[x_1, \ldots, x_k]$. Then, given

$$c[x_1, \ldots, x_k, x] \in C[x_1, \ldots, x_k, i(x_1, \ldots, x_k, x)],$$

$$d[x_1, \ldots, x_k, y] \in C[x_1, \ldots, x_k, j(x_1, \ldots, x_k, y)],$$

we may introduce a $(k+1)$-ary function constant f whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}], \quad A[x_1, \ldots, x_k] + B[x_1, \ldots, x_k]$$
$$\text{and} \quad C[x_1, \ldots, x_k, z]$$

by the schema

$$\begin{cases} f(x_1, \ldots, x_k, i(x_1, \ldots, x_k, x)) \text{ conv } c[x_1, \ldots, x_k, x], \\ f(x_1, \ldots, x_k, j(x_1, \ldots, x_k, y)) \text{ conv } d[x_1, \ldots, x_k, y]. \end{cases}$$

+-conversion. If f is a function constant which has been introduced by this schema and

$$a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}],$$
$$a \in A[a_1, \ldots, a_k] \quad \text{and} \quad b \in B[a_1, \ldots, a_k],$$

then

$$\begin{cases} f(a_1, ..., a_k, i(a_1, ..., a_k, a)) \text{ conv } c[a_1, ..., a_k, a], \\ f(a_1, ..., a_k, j(a_1, ..., a_k, b)) \text{ conv } d[a_1, ..., a_k, b]. \end{cases}$$

2.7. *Rules for I*

I-reflection. If $A[x_1, ..., x_k]$ is an n^{th} order type symbol depending on the variables $x_1 \in A_1, ..., x_k \in A_k[x_1, ..., x_{k-1}]$, then we may introduce a $(k+2)$-ary function constant I whose arguments and value have type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}],$$
$$A[x_1, ..., x_k], \quad A[x_1, ..., x_k] \quad \text{and} \quad V_n.$$

For $a_1 \in A_1, ..., a_k \in A_k[a_1, ..., a_{k-1}]$, $a \in A[a_1, ..., a_k]$ and $b \in A[a_1, ..., a_k]$, it is sometimes convenient to write $I_{A[a_1, ..., a_k]}(a, b)$ or $a =_{A[a_1, ..., a_k]} b$ instead of $I(a_1, ..., a_k, a, b)$.

I-introduction. Associated with the identity relation on $A[x_1, ..., x_k]$, there is a $(k+1)$-ary function constant r whose arguments and value have type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}],$$
$$A[x_1, ..., x_k] \quad \text{and} \quad I(x_1, ..., x_k, x, x).$$

I-elimination. If $x_1 \in A_1, ..., x_k \in A_k[x_1, ..., x_{k-1}]$ and $x \in A[x_1, ..., x_k]$ are variables with the indicated type symbols and

$$c[x_1, ..., x_k] \in C[x_1, ..., x_k, x, x, r(x_1, ..., x_k, x)]$$

where $C[x_1, ..., x_k, x, y, z]$ is a type symbol which, in addition to the parameters $x_1, ..., x_k$, depends on the variables $x \in A[x_1, ..., x_k]$, $y \in A[x_1, ..., x_k, x]$ and $z \in I(x_1, ..., x_k, x, y)$, then we may introduce a $(k+3)$-any function constant f whose arguments and value have type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}], \quad A[x_1, ..., x_k], \quad A[x_1, ..., x_k],$$
$$I(x_1, ..., x_k, x, y) \quad \text{and} \quad C[x_1, ..., x_k, x, y, z]$$

by the schema

$$f(x_1, ..., x_k, x, x, r(x_1, ..., x_k, x)) \text{ conv } c[x_1, ..., x_k, x].$$

I-conversion. If f is a function constant which has been introduced by this schema and $a_1 \in A_1, ..., a_k \in A_k[a_1, ..., a_{k-1}]$ and $a \in A[a_1, ..., a_k]$, then

$$f(a_1, ..., a_k, a, a, r(a_1, ..., a_k, a)) \text{ conv } c[a_1, ..., a_k, a].$$

2.8. *Rules for N_n*

N_n-reflection. For $n = 0, 1, ..., N_n$ is a constant with type symbol V_0.

N_n-introduction. $1, ..., n$ are constants with type-symbol N_n.

N_n-elimination. If

$$c_1[x_1, ..., x_k] \in C[x_1, ..., x_k, 1], ..., c_n[x_1, ..., x_k] \in C[x_1, ..., x_k, n]$$

where $C[x_1, ..., x_k, z]$ is a type symbol which depends not only on the parameters $x_1 \in A_1, ..., x_k \in A_k[x_1, ..., x_{k-1}]$ but also on the variable $z \in N_n$, then we may introduce a $(k+1)$-ary function constant f whose arguments and value have type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}], \quad N_n \quad \text{and} \quad C[x_1, ..., x_k, z]$$

by the schema

$$\begin{cases} f(x_1, ..., x_k, 1) \text{ conv } c_1[x_1, ..., x_k], \\ \vdots \\ f(x_1, ..., x_k, n) \text{ conv } c_n[x_1, ..., x_k]. \end{cases}$$

N_n-conversion. If the function constant f has been introduced by this schema and $a_1 \in A_1, ..., a_k \in A_k[a_1, ..., a_{k-1}]$, then

$$\begin{cases} f(a_1, ..., a_k, 1) \text{ conv } c_1[a_1, ..., a_k], \\ \vdots \\ f(a_1, ..., a_k, n) \text{ conv } c_n[a_1, ..., a_k]. \end{cases}$$

2.9. *Rules for N*

N-reflection. N is a constant with type symbol V_0.

N-introduction. 0 is a constant with type symbol N, and s is a unary function constant whose argument and value both have type symbol N.

N-elimination. Suppose that $x_1 \in A_1, ..., x_k \in A_k[x_1, ..., x_{k-1}]$, $x \in N$ and $y \in C[x_1, ..., x_k, x]$ are variables with the indicated type symbols where $C[x_1, ..., x_k, x]$ is a type symbol which depends not only on the parameters $x_1, ..., x_k$ but also on the numerical variable x. Then, given $c[x_1, ..., x_k] \in$

$C[x_1, ..., x_k, 0]$ and $d[x_1, ..., x_k, x, y] \in C[x_1, ..., x_k, s(x)]$, we may introduce a $(k+1)$-ary function constant f whose arguments and value have type symbols

$$A_1, ..., A_k[x_1, ..., x_{k-1}], \quad N \quad \text{and} \quad C[x_1, ..., x_k, x]$$

by the *recursion* schema

$$\begin{cases} f(x_1, ..., x_k, 0) \text{ conv } c[x_1, ..., x_k], \\ f(x_1, ..., x_k, s(x)) \text{ conv } d[x_1, ..., x_k, x, f(x_1, ..., x_k, x)]. \end{cases}$$

N-conversion. If the function constant f has been introduced by this schema and $a_1 \in A_1, ..., a_k \in A_k[a_1, ..., a_{k-1}]$ and $a \in N$, then

$$\begin{cases} f(a_1, ..., a_k, 0) \text{ conv } c[a_1, ..., a_k], \\ f(a_1, ..., a_k, s(a)) \text{ conv } d[a_1, ..., a_k, a, f(a_1, ..., a_k, a)]. \end{cases}$$

2.10. *Rules for V_n*

V_n-reflection. For every $n = 0, 1, ...$, V_n is a constant with type symbol V_{n+1}.

V_n-introduction. The reflection principles for the various basic types and type forming operations, which may be considered as the introduction rules for the universes V_n, have already been given.

There are no elimination rules and hence no rules of conversion for the universes V_n.

2.11. *Rules of conversion*

In addition to the rules of conversion associated with the basic types and type forming operations, which have been listed under these and correspond to the informal definitional schemata, there are the following general rules of conversion.

If a conv c and b conv d, where a and c are terms with type symbol $A[a_1, ..., a_k]$ and b and d are terms with type symbol $(\Pi x \in A[a_1, ..., a_k])B[a_1, ..., a_k, x]$, then $b(a)$ conv $d(c)$.

If f is a function constant whose arguments have type symbols $A_1, ..., A_k[x_1, ..., x_{k-1}]$ and a_1 conv $c_1, ..., a_k$ conv c_k, where $a_1 \in A_1, ..., a_k \in A_k[a_1, ..., a_{k-1}]$ and $c_1 \in A_1, ..., c_k \in A_k[c_1, ..., c_{k-1}]$, then $f(a_1, ..., a_k)$ conv $f(c_1, ..., c_k)$.

The above two rules of conversion can be condensed into the single rule which says that, if a_1 conv $c_1, ..., a_k$ conv c_k, then $b[a_1, ..., a_k]$ conv $b[c_1, ..., c_k]$. Here the type symbols of the terms $a_1, ..., a_k$ and $c_1, ..., c_k$ are as in the previous paragraph and $b[x_1, ..., x_k]$ is a term which depends on no other variables than $x_1 \in A_1, ..., x_k \in A_k[x_1, ..., x_{k-1}]$. There now only remain the rules of reflexivity, symmetry and transitivity.

If a is a term, that is, if $a \in A$ has been derived for some A, then a conv a.

If a conv b, then b conv a.

If a conv b and b conv c, then a conv c.

2.12. *Conversion of a type symbol*

If $a \in A$ and A conv B, then $a \in B$. This formalizes the principle of replacing the type of an object by a definitionally equal type.

One may ask if the epsilon relation ought not to be postulated to be compatible with convertibility to the left, that is, if a conv b and $b \in B$ should not imply $a \in B$. In the next chapter, we shall see that this rule actually holds as a derived rule of term formation. If it were included among the primitive rules, then it would be immediately clear that, parallel to a derivation of $a \in A$, there runs a derivation of $A \in V_n$ for some n. Indeed, when following the derivation of $a \in A$, the only rule that causes trouble is the rule

$$\frac{a \in A \qquad A \text{ conv } B}{a \in B}$$

By induction hypothesis, we then know that $A \in V_n$ for some n. Hence, if the epsilon relation were postulated to be compatible with convertibility to the left, we could immediately conclude $B \in V_n$ as desired.

2.13. *Axiom of choice*

Let x and y be variables with type symbols A and $B[x]$, respectively, and let $C[x, y]$ be a type symbol. We shall show how to derive the axiom of choice, that is, how to construct a closed term with type symbol

$$(\Pi x \in A)(\Sigma y \in B[x])C[x, y] \rightarrow$$
$$(\Sigma f \in (\Pi x \in A)B[x])(\Pi x \in A)C[x, f(x)].$$

In general, the type symbols $A, B[x]$ and $C[x, y]$ may depend on an arbitrary finite number of parameters in addition to the indicated variables, but it will be sufficient to consider the case without parameters.

To begin with, we define the left and right projections p and q on

$(\Sigma\, y \in B[x])C[x, y]$ by putting

$$\begin{cases} p(x, (x, y, z)) \text{ conv } y, \\ q(x, (x, y, z)) \text{ conv } z. \end{cases}$$

The elimination rule for Σ allows us to do this. Now, suppose

$$x \in A \quad \text{and} \quad z \in (\Pi\, x \in A)(\Sigma\, y \in B[x])C[x, y].$$

Then, by the rule of application,

$$z(x) \in (\Sigma\, y \in B[x])C[x, y]$$

and hence

$$p(x, z(x)) \in B[x] \qquad q(x, z(x)) \in C[x, p(x, z(x))].$$

Next, the schema of abstraction allows us to define a unary function constant f whose value has type symbol $(\Pi\, x \in A)B[x]$ by putting

$$f(z)(x) \text{ conv } p(x, z(x)).$$

This implies

$$C[x, p(x, z(x))] \text{ conv } C[x, f(z)(x)]$$

and hence we can convert the type symbol of $q(x, z(x))$ so as to obtain

$$q(x, z(x)) \in C[x, f(z)(x)].$$

Using abstraction again, we can define a unary function constant g whose value has type symbol $(\Pi\, x \in A)C[x, f(z)(x)]$ by putting

$$g(z)(x) \text{ conv } q(x, z(x)).$$

Pairing now shows that

$$(f(z), g(z)) \in (\Sigma\, f \in (\Pi\, x \in A)B[x])(\Pi\, x \in A)C[x, f(x)].$$

Hence, if we define the constant h by the abstraction schema

$$h(z) \text{ conv } (f(z), g(z)),$$

we obtain a closed term (in fact, a constant) whose type symbol is precisely the axiom of choice.

Note that this formal derivation of the axiom of choice is faithful in every step to the corresponding informal proof. (That the axiom of choice is implied by the very meaning of the intuitionistic notion of existence is pointed out, for instance, in [1].)

3. The model of closed normal terms, the normalization theorem (for closed terms) and its consequences

3.1. *Reduction*

The rules of *reduction* are the same as the rules of conversion, except that the symmetry rule (a conv b implies b conv a) is left out. If a reduces to b, we shall write

$$a \text{ red } b.$$

Observe that

$$\frac{a \in A \qquad A \text{ conv } B}{a \in B}$$

counts as a rule of term formation and not a rule of conversion, which means that, in it, conv should not be replaced by red. Reduction is the formal counterpart of the process of successively replacing definienda by their definientia (but not vice versa) in a mathematical expression.

3.2. *Closed normal terms*

Call a function constant an *introductory constant* if it has been introduced by means of one of the introduction rules, counting the reflection principles for the various types and type forming operations as introduction rules for the universes V_n. Note that the type symbols of the arguments and value of an introductory constant are always of the form

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad F(x_1, \ldots, x_k),$$

where F is an introductory type constant, denoting one of the basic types or type forming operations.

We shall say that c is a *closed normal term with type symbol* C if $c \in C$ can be derived by repeated applications of the following single rule of term formation. If f is a (possibly 0-ary) introductory constant whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad F(x_1, \ldots, x_k)$$

and $c_1 \in C_1, \ldots, c_k \in C_k$ are closed normal terms with the indicated type symbols and, furthermore, we have closed derivations of

$$A_1 \text{ red } C_1, \ldots, A_k[c_1, \ldots, c_{k-1}] \text{ red } C_k,$$

then $f(c_1, \ldots, c_k)$ is a closed normal term with type symbol $F(c_1, \ldots, c_k)$. Note that, if c is a closed normal term with type symbol C, then C is a *closed normal type symbol* (that is, a closed normal term with type

symbol V_n for some n) which is uniquely associated with c. If c is a closed normal term with type symbol C for some C, then c is a *closed normal term.*

Remembering what are the possible introductory constants, we see that a closed normal term must have one of the forms

$$(\Pi x \in A[a_1, ..., a_k])B[a_1, ..., a_k, x],$$
$$(\lambda x)b[a_1, ..., a_k, x],$$
$$(\Sigma x \in A[a_1, ..., a_k])B[a_1, ..., a_k, x],$$
$$(a_1, ..., a_k, a, b),$$
$$A[a_1, ..., a_k] + B[a_1, ..., a_k],$$
$$i(a_1, ..., a_k, a), \qquad j(a_1, ..., a_k, b),$$
$$I(a_1, ..., a_k, a, b),$$
$$r(a_1, ..., a_k, a),$$
$$N_n,$$
$$1, ..., n,$$
$$N,$$
$$s(s(... s(0) ...)),$$
$$V_n,$$

the constituent terms $a_1, ..., a_k, a$ and b also being closed and normal. A closed normal term with type symbol N, which obviously must have the form

$$s(s(... s(0) ...)),$$

is called a *numeral.*

Usually, one first defines a term a to be *normal* if it does not reduce to any term but itself, that is, if a red b implies that b is identical with a. Then the closed normal terms are defined to be those terms which are both closed and normal. Clearly, a closed normal term as defined above is a term which is both closed and normal, but the proof of the converse requires the Church–Rosser property which we shall be able to prove only as a corollary to the construction of the model of closed normal terms. Hence, in systems with a sufficiently complicated type structure, at least, the natural procedure is to define the closed normal terms inductively as done above.

3.3. **Normalization theorem.** *Every closed term reduces to a closed normal term*

The normalization theorem can be strengthened in two ways. First, it can be made to cover open terms as well, and, second, it can be proved that every (and not only some) reduction sequence starting from an arbitrary term leads to a unique normal term after a finite number of steps. The latter property is called strong normalization in Prawitz[17]. However, since these refinements cause certain technical complications in the proof which only obscure its main idea, we shall prove the normalization theorem in its simplest and most natural form.

The proof uses an extension of the method of convertibility introduced by Tait[22] in his proof of normalization for the terms of Gödel's[9] theory of primitive recursive functionals of finite type and systematically exploited in the Proceedings of the Second Scandinavian Logic Symposium. In Gödel's theory, the type symbols and the terms are generated separately from each other. This makes it possible, first, to define by induction on the construction of a type symbol the notion of convertibility for terms with that type symbol, and, second, to prove by induction on the construction of a term that it is convertible. In the present theory, however, the definition of the notion of convertibility and the proof that an arbitrary term is convertible can no longer be separated, because the type symbols and the terms are generated simultaneously. Instead, we shall show by induction on the length of a closed derivation, if it ends with $a \in A$, how to define a' and a'', where

> a' is a closed normal term with type symbol A', called the *normal form* of a, such that a red a',

and

> a'' is a proof of $A''(a')$, which it is sometimes more natural to think of as an object of type $A''(a')$,

and, if it ends with a conv b, that

$$a' =_{\text{def}} b' \quad \text{and} \quad a'' =_{\text{def}} b''.$$

More explicitly, remembering the interpretation of the notion of such that, this means that we associate with every closed term a three objects: a', a'' and the proof that a red a'. However, we shall try to manage without introducing an explicit notation for this proof. For the term V_n we

shall put, in particular,

$$V'_n =_{\text{def}} V_n$$

and

$V''_n =_{\text{def}}$ the function which to a closed normal term A with type symbol V_n assigns the type of n^{th} order species of closed normal terms with type symbol A.

Hence, for a closed type symbol A of order n, A' is a closed normal type symbol of the same order such that A red A', and A'' is an n^{th} order species of closed normal terms with type symbol A'. The normalization theorem is of course a consequence of the construction just outlined, because, for an arbitrary closed term a, a' is a closed normal term and a red a'.

In the course of the proof, we shall have to consider open derivations as well, that is, derivations which depend on certain variables $x_1 \in A_1, \ldots, x_k \in A_k[x_1, \ldots, x_{k-1}]$. For such a derivation, we shall show, if it ends with $a[x_1, \ldots, x_k] \in A[x_1, \ldots, x_k]$, how to define

$$a[x_1, \ldots, x_k]' =_{\text{def}} a'[x'_1, x''_1, \ldots, x'_k, x''_k],$$
$$a[x_1, \ldots, x_k]'' =_{\text{def}} a''[x'_1, x''_1, \ldots, x'_k, x''_k],$$

where

$a'[x'_1, x''_1, \ldots, x'_k, x''_k]$ is a closed normal term with type symbol $A'[x'_1, x''_1, \ldots, x'_k, x''_k]$ such that $a[x'_1, \ldots, x'_k]$ red $a'[x'_1, x''_1, \ldots, x'_k, x''_k]$

and $a''[x'_1, x''_1, \ldots, x'_k, x''_k]$ is a proof of

$$A''[x'_1, x''_1, \ldots, x'_k, x''_k](a'[x'_1, x''_1, \ldots, x'_k, x''_k]),$$

and, if it ends with $a[x_1, \ldots, x_k]$ conv $b[x_1, \ldots, x_k]$, that

$$a'[x'_1, x''_1, \ldots, x'_k, x''_k] =_{\text{def}} b'[x'_1, x''_1, \ldots, x'_k, x''_k],$$
$$a''[x'_1, x''_1, \ldots, x'_k, x''_k] =_{\text{def}} b''[x'_1, x''_1, \ldots, x'_k, x''_k].$$

Also, in case the derivation ends with $a[x_1, \ldots, x_k] \in A[x_1, \ldots, x_k]$, we shall have to verify the *substitution property*, namely, that, if we have derivations of $a_1 \in A_1, \ldots, a_k \in A_k[a_1, \ldots, a_{k-1}]$ to which the induction hypothesis is already applicable, then

$$a[a_1, \ldots, a_k]' =_{\text{def}} a'[a'_1, a''_1, \ldots, a'_k, a''_k],$$
$$a[a_1, \ldots, a_k]'' =_{\text{def}} a''[a'_1, a''_1, \ldots, a'_k, a''_k].$$

Observe that the square brackets in $a'[x'_1, x''_1, \ldots, x'_k, x''_k]$ and $a''[x'_1, x''_1, \ldots, x'_k, x''_k]$ are used in order to indicate that these expressions depend on the (informal!) variables $x'_1, x''_1, \ldots, x'_k, x''_k$ of which

x'_1 denotes an arbitrary closed normal term with type symbol A'_1,

x''_1 denotes an arbitrary proof of $A''_1(x'_1)$,

$\vdots$

x'_k denotes an arbitrary closed normal term with type symbol $A'_k[x'_1, x''_1, \ldots, x'_{k-1}, x''_{k-1}]$,

x''_k denotes an arbitrary proof of $A''_k[x'_1, x''_1, \ldots, x'_{k-1}, x''_{k-1}](x'_k)$.

It now only remains to go through systematically all the rules of term formation and conversion. When doing this, we shall always consider the case of no parameters in order to alleviate the notational burden.

Variables. When we make an assumption $x \in A$, we can apply the induction hypothesis to the derivation of $A \in V_n$. Hence we know that A' has been defined and is a closed normal term with type symbol $V'_n =_{\text{def}} V_n$ and that A'' has been defined and is an n^{th} order species of closed normal terms with type symbol A'. Therefore, we can let x' be an arbitrary closed normal term with type symbol A' and x'' an arbitrary proof of $A''(x')$.

Constants. With a function constant f whose arguments and value have type symbols

$$A_1, \ldots, A_k[x_1, \ldots, x_{k-1}] \quad \text{and} \quad A[x_1, \ldots, x_k],$$

we shall associate two functions f' and f'' of twice as many arguments $x'_1, x''_1, \ldots, x'_k, x''_k$ whose types are as above. For these arguments, $f'(x'_1, x''_1, \ldots, x'_k, x''_k)$ is to be a closed normal term with type symbol $A'[x'_1, x''_1, \ldots, x'_k, x''_k]$ such that

$$f(x'_1, \ldots, x'_k) \text{ red } f'(x'_1, x''_1, \ldots, x'_k, x''_k),$$

and $f''(x'_1, x''_1, \ldots, x'_k, x''_k)$ is to be a proof of

$$A''[x'_1, x''_1, \ldots, x'_k, x''_k](f'(x'_1, x''_1, \ldots, x'_k, x''_k)).$$

Hence, corresponding to an application of the rule of term formation

$$\frac{a_1 \in A_1 \cdots a_k \in A_k[a_1, \ldots, a_{k-1}]}{f(a_1, \ldots, a_k) \in A[a_1, \ldots, a_k]}$$

we can put

$$f(a_1, ..., a_k)' =_{\text{def}} f'(a_1', a_1'', ..., a_k', a_k''),$$
$$f(a_1, ..., a_k)'' =_{\text{def}} f''(a_1', a_1'', ..., a_k', a_k'').$$

Using the induction hypothesis and the substitution property, we see that $f(a_1, ..., a_k)'$ so defined is a closed normal term with type symbol $A[a_1, ..., a_k]'$ such that

$$f(a_1, ..., a_k) \text{ red } f(a_1', ..., a_k') \text{ red } f'(a_1', a_1'', ..., a_k', a_k'') =_{\text{def}} f(a_1, ..., a_k)'$$

and that $f(a_1, ..., a_k)''$ is a proof of $A[a_1, ..., a_k]''(f(a_1, ..., a_k)')$. And, obviously, the substitution property carries over from $a_1, ..., a_k$ to $f(a_1, ..., a_k)$.

Π-*reflection.* We put

$$(\Pi x \in A)B[x]' =_{\text{def}} (\Pi x \in A)B[x]$$

and, for a closed normal term c with type symbol $(\Pi x \in A)B[x]'$,

$(\Pi x \in A)B[x]''(c) =_{\text{def}}$ for all closed normal terms x' with type symbol A' and for all proofs x'' of $A''(x')$, $c(x')$ reduces to a closed normal term y' with type symbol $B'[x', x'']$ such that $B''[x', x''](y')$.

Clearly, if $A \in V_m$ and $B[x] \in V_n$, $(\Pi x \in A)B[x]'$ so defined is a closed normal type symbol of order $\max(m, n)$ such that

$$(\Pi x \in A)B[x] \text{ red } (\Pi x \in A)B[x]',$$

and $(\Pi x \in A)B[x]''$ is a species of closed normal terms with type symbol $(\Pi x \in A)B[x]'$ of the same order.

Π-*introduction.* If $b[x] \in B[x]$ depends on the variable $x \in A$ and the constant f with type symbol $(\Pi x \in A)B[x]$ has been introduced by abstraction,

$$f(x) \text{ conv } b[x],$$

we put

$$f' =_{\text{def}} f$$

and

$f'' =_{\text{def}}$ the function which to a closed normal term x' with type symbol A' and a proof x'' of $A''(x')$ assigns the triple consisting of $b'[x', x'']$, $b''[x', x'']$ and the proof of $f(x') \text{ red } b'[x', x'']$ which we get by transitivity from $f(x') \text{ red } b[x']$ and the proof of $b[x'] \text{ red } b'[x', x'']$ which we have constructed by induction hypothesis.

Clearly, f' so defined is a closed normal term with type symbol $(\Pi x \in A)B[x]'$ such that $f \text{ red } f'$, and f'' is a proof of $(\Pi x \in A)B[x]''(f')$.

Π-*elimination.* For $a \in A$ and $c \in (\Pi x \in A)B[x]$, we put

$$c(a)' =_{\text{def}} p'(c''(a', a'')),$$
$$c(a)'' =_{\text{def}} p''(c''(a', a'')),$$

where p' and p'' are the projections which take a proof of the existential proposition

there exists a closed normal term y' with type symbol $B'[a', a'']$ such that $B''[a', a''](y')$ and $c'(a') \text{ red } y'$

into the object y' and the proof of $B''[a', a''](y')$, respectively. By the substitution property, $B'[a', a''] =_{\text{def}} B[a]'$ and $B''[a', a''] =_{\text{def}} B[a]''$. Hence $c(a)'$ is a closed normal term with type symbol $B[a]'$ and $c(a)''$ is a proof of $B[a]''(c(a)')$. Also, from $a \text{ red } a'$, $c \text{ red } c'$ and $c'(a') \text{ red } c(a)'$, we can conclude $c(a) \text{ red } c(a)'$. That the substitution property carries over from a and c to $c(a)$ is obvious.

Π-*conversion.* If $a \in A$ and f has been introduced by the above schema of abstraction, then

$$f(a)' =_{\text{def}} p'(f''(a', a'')) =_{\text{def}} b'[a', a''] =_{\text{def}} b[a]',$$
$$f(a)'' =_{\text{def}} p''(f''(a', a'')) =_{\text{def}} b''[a', a''] =_{\text{def}} b[a]'',$$

as desired. Here the last steps in the two chains of definitional equalities follow from the substitution property.

Σ-*reflection.* Put

$$(\Sigma x \in A)B[x]' =_{\text{def}} (\Sigma x \in A)B[x]$$

and define the species $(\Sigma x \in A)B[x]''$ of closed normal terms with type symbol $(\Sigma x \in A)B[x]'$ by the proof condition

if x' is a closed normal term with type symbol A', x'' a proof of $A''(x')$, y' a closed normal term with type symbol $B'[x', x'']$ and y'' a proof of $B''[x', x''](y')$, then the quadruple (x', x'', y', y'') is a proof of $(\Sigma x \in A)B[x]''((x', y'))$.

Clearly, if $A \in V_m$ and $B[x] \in V_n$, then $(\Sigma\, x \in A)B[x]'$ so defined is a closed normal type symbol of order $\max(m, n)$ such that $(\Sigma\, x \in A)B[x]$ red $(\Sigma\, x \in A)B[x]'$, and $(\Sigma\, x \in A)B[x]''$ is a $\max(m, n)^{\text{th}}$ order species of closed normal terms with that type symbol.

Σ-introduction. For $x \in A$ and $y \in B[x]$, we put

$$(x, y)' =_{\text{def}} (x', y'),$$

which is a closed normal term with type symbol $(\Sigma\, x \in A)B[x]'$ such that (x', y') red $(x, y)'$, and

$$(x, y)'' =_{\text{def}} (x', x'', y', y''),$$

which is a proof of $(\Sigma\, x \in A)B[x]''((x, y)')$ as required.

Σ-elimination. If $c[x, y] \in C[(x, y)]$ depends on the variables $x \in A$ and $y \in B[x]$ and the unary function constant f has been introduced by the schema

$$f((x, y)) \text{ conv } c[x, y],$$

we put

$$f'((x', y'), (x', x'', y', y'')) =_{\text{def}} c'[x', x'', y', y''],$$
$$f''((x', y'), (x', x'', y', y'')) =_{\text{def}} c''[x', x'', y', y''].$$

By induction hypothesis, we know that $c'[x', x'', y', y'']$ is a closed normal term with type symbol

$$C[(x, y)]' =_{\text{def}} C'[(x, y)', (x, y)''] =_{\text{def}} C'[(x', y'), (x', x'', y', y'')]$$

such that $c[x', y']$ red $c'[x', x'', y', y'']$ and that $c''[x', x'', y', y'']$ is a proof of

$$C''[(x', y'), (x', x'', y', y'')](c'[x', x'', y', y''])$$
$$=_{\text{def}} C''[(x', y'), (x', x'', y', y'')](f'((x', y'), (x', x'', y', y''))).$$

(Note the tacit use of the principle that a proof of a proposition is also a proof of a definitionally equal proposition!) Hence the functions f' and f'' as we have defined them take a closed normal term z' with type symbol $(\Sigma\, x \in A)B[x]'$ and a proof z'' of $(\Sigma\, x \in A)B[x]''(z')$ into a closed normal term with type symbol $C'[z', z'']$ such that $f(z')$ red $f'(z', z'')$ and a proof of $C''[z', z''](f'(z', z''))$, respectively.

Σ-conversion. If $a \in A$ and $b \in B[a]$ and f has been introduced by the above schema, then

$$f((a, b))' =_{\text{def}} f'((a, b)', (a, b)'') =_{\text{def}} f'((a', b'), (a', a'', b', b''))$$
$$=_{\text{def}} c'[a', a'', b', b''] =_{\text{def}} c[a, b]',$$
$$f((a, b))'' =_{\text{def}} f''((a, b)', (a, b)'') =_{\text{def}} f''((a', b'), (a', a'', b', b''))$$
$$=_{\text{def}} c''[a', a'', b', b''] =_{\text{def}} c[a, b]'',$$

as desired. The last steps in the two chains of definitional equalities follow from the substitution property.

+-reflection. For $A \in V_m$ and $B \in V_n$, we put

$$(A + B)' =_{\text{def}} (A + B)$$

and define the species $(A + B)''$ of closed normal terms with type symbol $(A + B)'$ by the proof conditions

> if x' is a closed normal term with type symbol A' and x'' is a proof of $A''(x')$, then $i''(x', x'')$ is a proof of $(A + B)''(i(x))$,
>
> if y' is a closed normal term with type symbol B' and y'' is a proof of $B''(y')$, then $j''(y', y'')$ is a proof of $(A + B)''(j(y'))$.

Remembering that $A + B$ is but an informal notation for a constant with type symbol $V_{\max(m, n)}$, it is clear that $(A + B)'$ is a closed normal term with that type symbol such that $(A + B)$ red $(A + B)'$, and that $(A + B)''$ is a $\max(m, n)^{\text{th}}$ order species of closed normal terms with type symbol $(A + B)'$.

+-introduction. For a closed normal term x' with type symbol A' and a proof x'' of $A''(x')$, we put

$$i'(x', x'') =_{\text{def}} i(x'),$$

which is a closed normal term with type symbol $(A + B)'$ such that $i(x')$ red $i'(x', x'')$, and let i'' be the function introduced in the previous paragraph which takes x' and x'' into a proof of $(A + B)''(i(x'))$. The second rule of +-introduction is treated similarly.

+-elimination. If $c[x] \in C[i(x)]$ and $d[y] \in C[j(y)]$ depend on the variables $x \in A$ and $y \in B$, respectively, and the unary function constant f has been introduced by the schema

$$\begin{cases} f(i(x)) \text{ conv } c[x], \\ f(j(y)) \text{ conv } d[y], \end{cases}$$

we put

$$\begin{cases} f'(i(x'), i''(x', x'')) =_{\text{def}} c'[x', x''], \\ f'(j(y'), j''(y', y'')) =_{\text{def}} d'[y', y''], \end{cases}$$

$$\begin{cases} f''(i(x'), i''(x', x'')) =_{\text{def}} c''[x', x''], \\ f''(j(y'), j''(y', y'')) =_{\text{def}} d''[y', y'']. \end{cases}$$

Since

$$C[i(x)]' =_{\text{def}} C'[i(x'), i''(x', x'')],$$

$$C[i(x)]''(c[x]') =_{\text{def}} C''[i(x'), i''(x', x'')](f'(i(x'), i''(x', x''))),$$

$$f(i(x')) \text{ red } c[x'] \text{ red } c'[x', x''] =_{\text{def}} f'(i(x'), i''(x', x'')),$$

and correspondingly for $j(y)$ instead of $i(x)$, the functions f' and f'' take a closed normal term z' with type symbol $(A + B)'$ and a proof z'' of $(A + B)''(z')$ into a closed normal term with type symbol $C'[z', z'']$ such that $f(z')$ red $f'(z', z'')$ and a proof of $C''[z', z''](f'(z', z''))$, respectively.

+-conversion. If $a \in A$ and f has been introduced by the above schema, then

$$\begin{aligned} f(i(a))' &=_{\text{def}} f'(i(a)', i(a)'') =_{\text{def}} f'(i(a'), i''(a', a'')) \\ &=_{\text{def}} c'[a', a''] =_{\text{def}} c[a]', \\ f(i(a))'' &=_{\text{def}} f''(i(a)', i(a)'') =_{\text{def}} f''(i(a'), i''(a', a'')) \\ &=_{\text{def}} c''[a', a''] =_{\text{def}} c[a]'', \end{aligned}$$

as desired. Similarly for $j(b)$ instead of $i(a)$.

I-reflection. If the binary function constant I denotes the identity relation on $A \in V_n$, we put

$$I'(x', x'', y', y'') =_{\text{def}} I(x', y')$$

and define the species $I''(x', x'', y', y'')$ of closed normal terms with type symbol $I'(x', x'', y', y'')$ by the proof condition

> if x' is a closed normal term with type symbol A' and x'' is a proof of $A''(x')$, then $r''(x', x'')$ is a proof of $I''(x', x'', x', x'')(r(x'))$.

Clearly, I' and I'' so defined take closed normal terms x' and y' with type

symbol A' and proofs x'' and y'' of $A''(x')$ and $A''(y')$ into a closed normal term with type symbol V_n such that $I(x', y')$ red $I'(x', x'', y', y'')$ and an n^{th} order species of closed normal terms with type symbol $I'(x', x'', y', y'')$, respectively.

I-introduction. For a closed normal term x' with type symbol A' and a proof x'' of $A''(x')$, we put

$$r'(x', x'') =_{\text{def}} r(x'),$$

which is a closed normal term with type symbol $I'(x', x'', x', x'') =_{\text{def}} I(x', x')$ such that $r(x')$ red $r'(x', x'')$, and let r'' be the function introduced in the previous paragraph which takes x' and x'' into a proof of $I''(x', x'', x', x'')(r(x'))$.

I-elimination. If $c[x] \in C[x, x, r(x)]$ depends on the variable $x \in A$ and the function constant f has been introduced by the schema

$$f(x, x, r(x)) \text{ conv } c[x],$$

we put

$$\begin{aligned} f'(x', x'', x', x'', r(x'), r''(x', x'')) &=_{\text{def}} c'[x', x''], \\ f''(x', x'', x', x'', r(x'), r''(x', x'')) &=_{\text{def}} c''[x', x'']. \end{aligned}$$

Since

$$C[x, x, r(x)]' =_{\text{def}} C'[x', x'', x', x'', r(x'), r''(x', x'')],$$

$$C[x, x, r(x)]''(c[x]') =_{\text{def}} C''[x', x'', x', x'', r(x'), r''(x', x'')] (f'(x', x'', x', x'', r(x'), r''(x', x''))),$$

$$f(x', x', r(x')) \text{ red } c[x'] \text{ red } c'[x', x''] =_{\text{def}} f'(x', x'', x', x'', r(x'), r''(x', x'')),$$

the functions f' and f'' take closed normal terms x' and y' with type symbol A', proofs x'' and y'' of $A''(x')$ and $A''(y')$, respectively, a closed normal term z' with type symbol $I'(x', x'', y', y'') =_{\text{def}} I(x', y')$ and a proof z'' of $I''(x', x'', y', y'')(z')$ into a closed normal term with type symbol $C'[x', x'', y', y'', z', z'']$ such that

$$f(x', y', z') \text{ red } f'(x', x'', y', y'', z', z'')$$

and a proof of $C''[x', x'', y', y'', z', z''](f'(x', x'', y', y'', z', z''))$, respectively.

I-conversion. If $a \in A$ and the function constant f has been introduced by the above schema, then

$$\begin{aligned} f(a, a, r(a))' &=_{\text{def}} f'(a', a'', a', a'', r(a'), r''(a', a'')) \\ &=_{\text{def}} c'[a', a''] =_{\text{def}} c[a]', \\ f(a, a, r(a))'' &=_{\text{def}} f''(a', a'', a', a'', r(a'), r''(a', a'')) \\ &=_{\text{def}} c''[a', a''] =_{\text{def}} c[a]'', \end{aligned}$$

as desired.

N_n-reflection. We put

$$N_n' =_{\text{def}} N_n$$

and define the species N_n'' of closed normal terms with type symbol N_n' by the proof condition

$$m'' \text{ is a proof of } N_n''(m) \text{ for } m = 1, \ldots, n.$$

Clearly, N_n' is a closed normal term with type symbol $V_0' =_{\text{def}} V_0$ such that N_n red N_n', and N_n'' is a 0^{th} order species of closed normal terms with type symbol N_n'.

N_n-introduction. For $m = 1, \ldots, n$, we put

$$m' =_{\text{def}} m,$$

which is a closed normal term with type symbol $N_n' =_{\text{def}} N_n$ such that m red m', and let m'' be the proof of $N_n''(m') =_{\text{def}} N_n''(m)$ which enters into the proof condition for N_n''.

N_n-elimination. If $c_1 \in C[1], \ldots, c_n \in C[n]$ and the function constant f has been introduced by the schema

$$\begin{cases} f(1) \text{ conv } c_1, \\ \quad \vdots \\ f(n) \text{ conv } c_n, \end{cases}$$

we put

$$\begin{cases} f'(1, 1'') =_{\text{def}} c_1', \\ \quad \vdots \\ f'(n, n'') =_{\text{def}} c_n', \end{cases}$$

$$\begin{cases} f''(1, 1'') =_{\text{def}} c_1'', \\ \quad \vdots \\ f''(n, n'') =_{\text{def}} c_n''. \end{cases}$$

Since

$$\begin{aligned} &C[m]' =_{\text{def}} C'[m, m''], \\ &C[m]''(c_m') =_{\text{def}} C''[m, m''](f'(m, m'')), \\ &f(m) \text{ red } c_m \text{ red } c_m' =_{\text{def}} f'(m, m''), \end{aligned}$$

for $m = 1, \ldots, n$, the functions f' and f'' take a closed normal term x' with type symbol $N_n' =_{\text{def}} N_n$ and a proof x'' of $N_n''(x')$ into a closed normal term with type symbol $C'[x', x'']$ such that $f(x')$ red $f'(x', x'')$ and a proof of $C''[x', x''](f'(x', x''))$, respectively.

N_n-conversion. If the function constant f has been introduced by the above schema, then

$$\begin{aligned} f(m)' &=_{\text{def}} f'(m', m'') =_{\text{def}} f'(m, m'') =_{\text{def}} c_m', \\ f(m)'' &=_{\text{def}} f''(m', m'') =_{\text{def}} f''(m, m'') =_{\text{def}} c_m'' \end{aligned}$$

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

N-reflection. We put

$$N' =_{\text{def}} N,$$

which is a closed normal term with type symbol $V_0' =_{\text{def}} V_0$ such that N red N', and define inductively the 0^{th} order species N'' of closed normal terms with type symbol $N' =_{\text{def}} N$ by the proof conditions

$$0'' \text{ is a proof of } N''(0)$$

and

> if x' is a closed normal term with type symbol N and x'' is a proof of $N''(x')$, then $s''(x', x'')$ is a proof of $N''(s(x'))$.

N-introduction. We put

$$0' =_{\text{def}} 0,$$

which is a closed normal term with type symbol $N' =_{\text{def}} N$ such that 0 red $0'$, and let $0''$ be the proof of $N''(0') =_{\text{def}} N''(0)$ which enters into the first proof condition for N''. Similarly, for a closed normal term x' with type

symbol $N' =_{\text{def}} N$ and a proof x'' of $N''(x')$, we put

$$s'(x', x'') =_{\text{def}} s(x'),$$

which is a closed normal term with type symbol $N' =_{\text{def}} N$ such that $s(x')$ red $s'(x', x'')$, and let s'' be the function which enters into the second proof condition for N'' and takes x' and x'' into a proof of $N''(s'(x', x'')) =_{\text{def}} N''(s(x'))$.

N-elimination. If $c \in C[0]$, $d[x, y] \in C[s(x)]$ depends on the variables $x \in N$ and $y \in C[x]$, and the function constant f has been introduced by recursion,

$$\begin{cases} f(0) \text{ conv } c, \\ f(s(x)) \text{ conv } d[x, f(x)], \end{cases}$$

we define f' and f'' by the simultaneous recursions

$$\begin{cases} f'(0, 0'') =_{\text{def}} c', \\ f'(s(x'), s''(x', x'')) =_{\text{def}} d'[x', x'', f'(x', x''), f''(x', x'')], \end{cases}$$

$$\begin{cases} f''(0, 0'') =_{\text{def}} c'', \\ f''(s(x'), s''(x', x'')) =_{\text{def}} d''[x', x'', f'(x', x''), f''(x', x'')]. \end{cases}$$

Since

$$\begin{cases} C[0]' =_{\text{def}} C'[0, 0''], \\ C[s(x)]' =_{\text{def}} C'[s(x'), s''(x', x'')], \end{cases}$$

$$\begin{cases} C[0]''(c') =_{\text{def}} C''[0, 0''](f'(0, 0'')), \\ C[s(x)]''(d'[x', x'', f'(x', x''), f''(x', x'')]) \\ \quad =_{\text{def}} C''[s(x'), s''(x', x'')](f'(s(x'), s''(x', x''))), \end{cases}$$

$$f(0) \text{ red } c \text{ red } c' =_{\text{def}} f'(0, 0'')$$

and, under the induction hypothesis that $f(x')$ red $f'(x', x'')$,

$$f(s(x')) \text{ red } d[x', f(x')] \text{ red } d[x', f'(x', x'')]$$
$$\text{red } d'[x', x'', f'(x', x''), f''(x', x'')] =_{\text{def}} f'(s(x'), s''(x', x'')),$$

the functions f' and f'' take a closed normal term x' with type symbol $N' =_{\text{def}} N$ and a proof x'' of $N''(x')$ into a closed normal term with type symbol $C'[x', x'']$ such that $f(x')$ red $f'(x', x'')$ and a proof of $C''[x', x'']$ $(f'(x', x''))$, respectively.

N-conversion. If $a \in N$ and the function constant f has been introduced by recursion, then

$$\begin{cases} f(0)' =_{\text{def}} f'(0, 0'') =_{\text{def}} c', \\ f(s(a))' =_{\text{def}} f'(s(a'), s''(a', a'')) \\ \quad =_{\text{def}} d'[a', a'', f'(a', a''), f''(a', a'')] =_{\text{def}} d[a, f(a)]', \end{cases}$$

$$\begin{cases} f(0)'' =_{\text{def}} f''(0, 0'') =_{\text{def}} c'', \\ f(s(a))'' =_{\text{def}} f''(s(a'), s''(a', a'')) \\ \quad =_{\text{def}} d''[a', a'', f'(a', a''), f''(a', a'')] =_{\text{def}} d[a, f(a)]'', \end{cases}$$

as required.

V_n-reflection. We have already defined V_n' and V_n'', but we also have to verify that $V_n' =_{\text{def}} V_n$ is a closed normal term with type symbol $V_{n+1}' =_{\text{def}} V_{n+1}$ such that V_n red V_n', which is clearly so, and that V_n'' is an $(n+1)^{\text{th}}$ order species of closed normal terms with type symbol $V_n' =_{\text{def}} V_n$. Remember that, for a closed normal term A with type symbol V_n,

> $V_n''(A) =_{\text{def}}$ the type of n^{th} order species of closed normal terms with type symbol A, that is, the type of functions from the closed normal terms with type symbol A into the n^{th} universe,

which is an object of the $(n+1)^{\text{th}}$ universe. Hence V_n'' is indeed an $(n+1)^{\text{th}}$ order species of closed normal terms with type symbol $V_n' =_{\text{def}} V_n$.

This finishes the construction of the model of closed normal terms and thereby the proof of the normalization theorem for closed terms.

3.4. THEOREM. *If a and b are closed normal terms such that a conv b, then $a = b$, that is, a and b are syntactically identical.*

The following proof is due to Peter Hancock. From the construction of the term model, we know that a red a' and b red b'. But a closed normal

term can only reduce to itself, and hence $a = a'$ and $b = b'$. On the other hand, a conv b implies $a' =_{\text{def}} b'$ (and $a'' =_{\text{def}} b''$, but we do not need that) and, a fortiori, $a' = b'$. Hence $a = a' = b' = b$ as was to be proved.

3.5. THEOREM. *If the closed term a converts into the closed normal term b, then a reduces to b.*

We know from the normalization theorem that a red a' and, by assumption, that a conv b, where a' and b are both closed normal terms. From a red a' and a conv b, we can conclude a' conv b. Hence, by the previous theorem, $a' = b$ which, together with a red a', yields a red b as desired.

3.6. UNIQUENESS OF NORMAL FORM. *If the closed term a converts into the closed normal terms b and c, then $b = c$.*

The assumptions a conv b and a conv c imply b conv c and hence, b and c being closed normal terms, $b = c$.

3.7. CHURCH–ROSSER PROPERTY. *If a and b are closed terms, then a conv b if and only if there exists a closed term c such that a red c and b red c.*

For systems with a sufficiently simple type structure, like the typed lambda calculus or the system of Gödel terms, the Church–Rosser property can be proved by pure combinatorial means. For the present theory, however, it is an open question whether a combinatorial proof can at all be given. The following proof, by the Hancock method, uses in an essential way the properties of the model of closed normal terms.

We know that a red a' and b red b'. Moreover, a conv b implies $a' =_{\text{def}} b'$. Hence, if we put $c =_{\text{def}} a' =_{\text{def}} b'$, we can conclude a red c and b red c as desired.

3.8. DECIDABILITY OF THE CONVERTIBILITY RELATION. *For two closed terms a and b, it can be mechanically decided whether or not a* conv *b.*

This was first proved for the Gödel terms by Tait[22]. The (clearly mechanical) decision procedure is this: Reduce a and b to their normal forms a' and b' and check whether or not $a' = b'$, that is, whether or not a' and b' are syntactically identical.

3.9. UNIQUENESS OF TYPE SYMBOLS. *If a and b are closed terms with type symbols A and B, respectively, and a* conv *b, then A* conv *B*

From the properties of the term model, we know that

$$a \text{ red } a', \qquad A \text{ red } A',$$
$$b \text{ red } b', \qquad B \text{ red } B',$$

where a' and b' are closed normal terms with type symbols A' and B', respectively, and, since a conv b, that $a' =_{\text{def}} b'$. Hence, if we put $c =_{\text{def}} a' =_{\text{def}} b'$, c is at the same time a closed normal term with type symbol A' and a closed normal term with type symbol B'. However, as remarked in connection with the definition of the closed normal terms, the type symbol of a closed normal term is uniquely associated with it, and hence $A' = B'$ so that A conv B as was to be proved.

3.10. COROLLARY. *If there are closed derivations of $A \in V_m$, $B \in V_n$ and A* conv *B, then $m = n$*

This shows that the order of a closed type symbol is unique. From the previous theorem, we can conclude that V_m conv V_n. But V_m and V_n are both closed normal terms, and hence $V_m = V_n$, that is, $m = n$.

3.11. THEOREM. *From closed derivations of a* conv *b and $b \in B$, we can find a closed derivation of $a \in B$*

This shows that

$$\frac{a \text{ conv } b \qquad b \in B}{a \in B}$$

holds as a derived rule for closed derivations. To prove it, first find a closed derivation of $a \in A$ for some A by following the derivation of a conv b. Then, by the uniqueness of type symbols, we can find a closed derivation of A conv B. Applying the rule for converting a type symbol to these two derivations,

$$\frac{a \in A \qquad A \text{ conv } B}{a \in B}$$

we get the desired derivation of $a \in B$.

3.12. THEOREM. *From a closed derivation of $a \in A$, we can find a closed derivation of $A \in V_n$ for some n*

The proof is by induction on the length of the derivation of $a \in A$. The

only rule which causes any difficulty is the rule

$$\frac{a \in A \qquad A \text{ conv } B}{a \in B}$$

The induction hypothesis then allows us to find a closed derivation of $A \in V_n$ for some n. Hence, by the previous theorem and the symmetry of the convertibility relation, $B \in V_n$ for the same value of n.

3.13. DECIDABILITY OF THE EPSILON RELATION. *It can be mechanically decided whether or not a closed term is a type symbol. And, given a closed term a and a closed type symbol A, it can be mechanically decided whether or not $a \in A$*

A closed term is a type symbol if and only if the type symbol of its normal form is syntactically identical with V_n for some $n = 0, 1, \ldots$, and there is clearly a mechanical procedure for checking whether or not this is the case. Similarly, if a is a closed term and A a closed type symbol, then $a \in A$ if and only if the type symbol of the normal form a' of a is syntactically identical with the normal form A' of A, and again this is something that can be mechanically decided.

The decidability of the epsilon relation shows that the theory of types satisfies the adequacy condition formulated in (the discussion following)[15, Problem 10], namely, that it should be recursively decidable whether or not a closed term formally proves a given closed formula in the hypothetical theory of constructions. This is the formal counterpart of the experience that we can decide whether or not a purported proof actually is a proof of a given proposition (in Kreisel's words: we recognize a proof when we see one).

3.14. THEOREM. *$I(a, b)$ is provable, that is, there is a closed term with type symbol $I(a, b)$, if and only if a conv b*

Here, of course, it is supposed that a and b are closed terms with common type symbol A and that I denotes the identity relation on A. The sufficiency is trivial, because $r(a)$ is a closed term with type symbol $I(a, a)$ and a conv b implies $I(a, a)$ conv $I(a, b)$ so that, by the rule for converting a type symbol, $r(a)$ is a closed term with type symbol $I(a, b)$. Conversely, suppose that c is a closed term with type symbol $I(a, b)$. Then c' is a closed normal term with type symbol $I(a, b)' =_{\text{def}} I(a', b')$ which is only possible if $a' = b'$ and $r(a') = c'$. Since a red a', b red b' and $a' = b'$, we can conclude a conv b as desired.

3.15. THEOREM. *A number theoretic function which can be constructed in the theory of types is mechanically computable*

Of course, the fact that there is a not necessarily mechanical procedure for computing every function in the present theory requires no proof at all once we have recognized that the axioms and rules of inference of the theory are consonant with the intuitionistic notion of function, according to which a function is the same as a rule or method.

By saying that a number theoretic function can be constructed in the theory of types, I mean that there is a closed term f with type symbol $N \to N$ which denotes it. Suppose that we want to find the value of the function for a certain natural number which is denoted by the numeral m. Then $f(m)$ denotes the value of the function for this argument. But $f(m)$ is a closed term with type symbol N and hence, by the normalization theorem and the fact that the closed normal terms with type symbol N are precisely the numerals, it reduces to a numeral n. It only remains to remark that the normal form of a term can be found in a purely mechanical way, that is, by manipulating symbol strings according to rules which refer solely to their syntactical form and not to their meaning.

Similarly, having formalized the construction of the real numbers (for example, as Cauchy sequences of rationals) in the theory of types, we can prove as a corollary to the normalization theorem that every individual real number which we can construct in the formal theory can be computed by a machine with any preassigned degree of approximation.

These corollaries show that formalization taken together with the ensuing proof-theoretical analysis effectuates the computerization of abstract intuitionistic mathematics that above all Bishop[1] and [2] has asked for. What is doubtful at present is not whether computerization is possible, because we already know that, but rather whether these proof-theoretical computation procedures are at all useful in practice. So far, they do not seem to have found a single significant application.

References

[1] E. Bishop, *Foundations of Constructive Analysis* (McGraw-Hill, New York, 1967).

[2] E. Bishop, Mathematics as a numerical language, in: J. Myhill, A. Kino and R. E. Vesley, eds., *Intuitionism and Proof Theory* (North-Holland, Amsterdam, 1970) pp. 53–71.

[3] H. B. Curry and R. Feys, *Combinatory Logic*, Vol. I (North-Holland, Amsterdam, 1968).

[4] S. Feferman, Systems of predicative analysis, *Journal of Symbolic Logic* **29** (1964) 1–30.
[5] S. Feferman, Set-theoretical foundations of category theory, in: *Reports of the Midwest Category Theory Seminar* III, Lecture Notes in Mathematics, Vol. 106 (Springer, Berlin, 1969) pp. 201–247.
[6] G. Gentzen, Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift* **39** (1934) 176–210, 405–431.
[7] J.-Y. Girard, Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse, Université Paris VII (1972).
[8] N. D. Goodman, A theory of constructions equivalent to arithmetic, in: J. Myhill, A. Kino and R. E. Vesley, eds., *Intuitionism and Proof Theory* (North-Holland, Amsterdam, 1970) pp. 101–120.
[9] K. Gödel, Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes, *Dialectica* **12** (1958) 280–287.
[10] W. A. Howard, The formulae-as-types notion of construction (1969), Unpublished.
[11] W. A. Howard, A system of abstract constructive ordinals, *Journal of Symbolic Logic*, **37** (1972) 355–374.
[12] G. Kreisel, Foundations of intuitionistic logic, in: E. Nagel, P. Suppes and A. Tarski, eds., *Logic, Methodology and Philosophy of Science* (Stanford University Press, Stanford, Calif. 1962) pp. 198–210.
[13] G. Kreisel, Mathematical logic, in: T. L. Saaty, ed., *Lectures on Modern Mathematics*, Vol. III (Wiley, New York, 1965) pp. 95–195.
[14] G. Kreisel, Functions, ordinals, species, in B. van Rootselaar and J. F. Staal, eds., *Logic, Methodology and Philosophy of Science* III (North-Holland, Amsterdam, 1968) pp. 145–159.
[15] G. Kreisel, Church's thesis: a kind of reducibility axiom for constructive mathematics, in: J. Myhill, A. Kino and R. E. Vesley, eds., *Intuitionism and Proof Theory* (North-Holland, Amsterdam, 1970) pp. 121–150.
[16] P. Martin-Löf, About models for intuitionistic type theories and the notion of definitional equality, paper read at the Orléans Logic Conference (1972).
[17] D. Prawitz, Ideas and results in proof theory, in: *Proceedings of the Second Scandinavian Logic Symposium*, ed. J. E. Fenstad, (North-Holland, Amsterdam, 1971) pp. 235–307.
[18] B. Russell, *The Principles of Mathematics*, Vol. I (Cambridge University Press, Cambridge, 1903).
[19] D. Scott, Constructive validity, in: *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics, Vol. 125 (Springer, Berlin, 1970) pp. 237–275.
[20] K. Schütte, Predicative well-orderings, in: J. N. Crossley and M. Dummett, eds., *Formal Systems and Recursive Functions* (North-Holland, Amsterdam, 1963) pp. 279–302.
[21] K. Schütte, Eine Grenze für die Beweisbarkeit der Transfiniten Induktion in der verzweigten Typenlogik, *Archiv für mathematische Logik und Grundlagenforschung* 7 (1965) 45–60.
[22] W. W. Tait, Intentional interpretations of functionals of finite type, *Journal of Symbolic Logic* **32** (1967) 198–212.
[23] W. W. Tait, Constructive reasoning, in: B. van Rootselaar and J. F. Staal, eds., *Logic, Methodology and Philosophy of Science* III (North-Holland, Amsterdam, 1968) pp. 185–199.