An intuitionistic theory of types

Per Martin-Löf
Department of Mathematics, University of Stockholm

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 1967. The language of the theory is richer than the language of first order predicate logic. This makes it possible to strengthen the axioms for existence and disjunction. In the case of existence, the possibility of strengthening the usual elimination rule seems first to have been indicated by Howard 1969, whose proposed axioms are special cases of the existential elimination rule of the present theory. Furthermore, there is a reflection principle which links the generation of objects and types and plays somewhat the same role for the present theory as does the replacement axiom for Zermelo-Fraenkel set theory.

An earlier, not yet conclusive, attempt at formulating a theory of this kind was made by Scott 1970. Also related, although less closely, are the type and logic free theories of constructions of Kreisel 1962 and 1965 and Goodman 1970.

In its first version, the present theory was based on the strongly impredicative axiom that there is a type of all types whatsoever, which is at the same time a type and an object of that type. This axiom had to be abandoned, however, after it was shown to lead to a contradiction by Jean Yves Girard. I am very grateful to him for showing me his paradox. The change that it necessitated is so drastic that my theory no longer contains intuitionistic simple type theory as it originally did. Instead, its proof theoretic strength should be close to that of predicative analysis.

1. INFORMAL EXPLANATIONS OF THE BASIC CONCEPTS.

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 1903 *doctrine of types*, 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 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 1967. Put differently, a type is welldefined if we understand (or *grasp* to use a word favoured by Kreisel 1970) 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. For example

$$971 \text{ is a non prime number}$$

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 explanations of the logical operations when taken together with the doctrine of types. For example, the intuitionistic notion of implication is explained by saying that a proof of $A \supset B$ is a function which to an arbitrary proof of A assigns a proof of B. And, if every function is to have a type as its domain, this requires that the proofs of the proposition A must form a type. To avoid an unwieldy mode of expression and notation, I shall sometimes simply identify a proposition with the type that represents it. When a type A represents a proposition,

$$a \in A$$

may be read alternatively

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

On the formal level, the analogy between formulae and types was discovered by Curry and Feys 1958 and further extended by Howard 1969 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 1970.

1.3. Cartesian product of a family of types. Suppose now that A is a type and that B is a *function*, *rule* or *method* which to an arbitrary object a of type A assigns a type $B(a)$. Then the *cartesian product*

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

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

$$b(a)$$

of type $B(a)$. The notation $b(a_1, \ldots, a_n)$ will be preferred to $b(a_1) \ldots (a_n)$. When $B(a)$ represents a proposition for every object a 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 a of type A assigns a proof of $B(a)$.

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

$$(\lambda x)b[x](a) = b[a].$$

Here $b[a]$ denotes the result of substituting the object a of type A for the variable x in the term $b[x]$.

If $B(a)$ is defined to be one and the same type B for every object a 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. Parentheses are associated to the right so that $A_1 \to \ldots \to A_{n-1} \to A_n$ abbreviates $A_1 \to (\ldots (A_{n-1} \to A_n) \ldots)$. 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.

1.4. Disjoint union of a family of types. Given a type A and a function B which to an object a of type A assigns a type $B(a)$, we may form the *disjoint union*

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

which is the type of pairs (a, b) where a and b are objects of type A and $B(a)$, respectively. When $B(a)$ represents a proposition for every object a 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 (a, b) where a is an object of type A and b a proof of the proposition $B(a)$.

Let C be a function which to an arbitrary object of type $(\Sigma x \in A)B(x)$ assigns a type. Given a function d of type $(\Pi x \in A)(\Pi y \in B(x))C((x, y))$ we may then introduce a function of type $(\Pi z \in (\Sigma x \in A)B(x))C(z)$ whose value for the argument c will be denoted $E(c, d)$ by the schema

$$E((a, b), d) = d(a, b).$$

In particular, we can introduce the left and right projections p and q of types $(\Sigma x \in A)B(x) \to A$ and $(\Pi z \in (\Sigma x \in A)B(x))B(p(z))$, respectively, by putting

$$\begin{cases} p((a,b)) & = & a, \\ q((a,b)) & = & b. \end{cases}$$

A third function of $(\Sigma x \in A)B(x)$ is to represent

the type of all objects a of type A *such that* $B(a)$,

because, from the intuitionistic point of view, to give an object a of type A such that $B(a)$ is to give a *together* with a proof b of the proposition $B(a)$. This interpretation of the notion of such that is implicitly used by Bishop 1967 and discussed by Kreisel 1968. However, its explicit formulation requires us to consider 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 (a, b) where a is a sequence of rational numbers and b is a proof that a 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 c^{-1} of a non zero real number c depends effectively on the proof that $c \neq 0$. A similar phenomenon occurs in the intuitionistic theory of ordinals of the second number class (see Brouwer 1918) 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 $b - a$ of two ordinals a and b depends effectively on the proof that $a < b$.

In the special case when $B(a)$ is defined to be one and the same type B for every object a 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

$$A \,\&\, B.$$

1.5. 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 $i(a)$ with a of type A or $j(b)$ with b of type B. Here i and j denote the canonical injections. When A and B both represent propositions, $A + B$ represents their *disjunction*

$$A \vee B.$$

Let C be a function which to an arbitrary object of type $A + B$ assigns a type, and suppose that d and e are functions of types $(\Pi x \in A)C(i(x))$ and $(\Pi y \in B)C(j(y))$, respectively. Then we may define a function of type $(\Pi z \in A+B)C(z)$ whose value for the argument c will be denoted $D(c, d, e)$ by the schema

$$\begin{cases} D(i(a), d, e) &= d(a), \\ D(j(b), d, e) &= e(b). \end{cases}$$

1.6. 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 union of N_1 with itself n times.

If C is a function which to an arbitrary object 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 of type $(\Pi x \in N_n)C(x)$ whose value for the argument c will be denoted $R_n(c, c_1, \ldots, c_n)$ by the schema

$$\begin{cases} R_n(1, c_1, \ldots, c_n) &= c_1, \\ \qquad\vdots & \\ R_n(n, c_1, \ldots, c_n) &= c_n. \end{cases}$$

In particular, N_0 is the *empty type* $\emptyset$ which also represents the logical constant *falsehood* $\perp$, and the function $(\lambda x)R_0(x)$ of type $(\Pi x \in N_0)C(x)$ is the empty function. Similarly, the one element type N_1 is used to represent the logical constant *truth* $\top$.

1.7. Natural numbers. N is a type, namely, the type of *natural numbers*. 0 is an object of type N and, if n is an object of type N, so is its successor $s(n)$. 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 d of type $C(0)$ and a function e of type $(\Pi x \in N)(C(x) \to$

$C(s(x)))$, we may introduce a function of type $(\Pi x \in N)C(x)$ whose value for the argument n will be denoted $R(n,d,e)$ by the *recursion* schema

$$\begin{cases} R(0,d,e) = d, \\ R(s(n),d,e) = e(n,R(n,d,e)). \end{cases}$$

If $C(n)$ represents a proposition for every natural number n, then $(\lambda x)R(x,d,e)$ is the proof of the universal proposition $(\forall x \in N)\, C(x)$ which we get by applying the principle of mathematical *induction* to the proof d of $C(0)$ and the proof e of $(\forall x \in N)\,(C(x) \to C(s(x)))$

The type N is just the prime example of a type introduced by an *ordinary inductive definition*. However, it seems preferable to treat this special case rather than to give the necessarily much more complicated general formulation which would include $(\Sigma x \in A)B(x)$, $A + B$, N_n and N as special cases. See Martin-Löf 1971 for a general formulation of inductive definitions in the language of first order predicate logic.

1.8. Reflection principle. 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 equality between natural numbers by the schema

$$\begin{cases} E(0,0) = \top, \\ E(s(m),0) = \bot, \\ E(0,s(n)) = \bot, \\ E(s(m),s(n)) = E(m,n), \end{cases}$$

which will give us in particular the third and fourth Peano axioms. This can clearly be done by recursion if only the propositions alias types $\bot$ and $\top$ were objects of some type V. Also, we want to be able to define transfinite types like $(\Pi x \in N)F(x)$ where

$$\begin{cases} F(0) = N, \\ F(s(n)) = F(n) \to N. \end{cases}$$

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 under the following inductive clauses. N_0, N_1, ...and N are objects of type V. If A and B are objects of type V, then so is $A + B$. 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. Note, however, that the reflection principle does *not* justify the axiom that V is

an object of type V which Girard 1972 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 over 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 free to throw new types into it or require it to be closed with respect to new type forming operations. For example, we may want to introduce the type O of ordinals of the second number class or the operation which to a type A assigns the type $W(A)$ of wellfounded trees over A (see Tait 1968, Scott 1970 and Howard 1971).

Borrowing terminology from category theory, a type which is an object of V is said to be *small* whereas V itself and all types which are derived 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 theoretical strength, is adequate for the formulation of the basic notions and constructions of category theory. 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 finite types and, on the 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 theoretical reflection principle for the formalization of category theory is due to Kreisel 1965 and has been elaborated by Feferman 1969.

1.9. Girard's paradox. Suppose that we think of V not as the type of *small* types but as the type of *all* types whatsoever. Then, being a type, namely, the type of types, V is itself an object of type V, in short,

$$V \in V,$$

and a type is the same as an object of type V. The following paradox which is a modification of the one discovered by Girard 1972 (which, in turn, resembles the Burali-Forti paradox) shows that the idea of the type of all types whatsoever is inconsistent.

Define an *ordering without infinite descending chains* (Girard 1972 introduces instead what he calls *torsion free orderings*) to be a type A together with a binary relation $<$ on A such that the propositions

$$P(A,<) = (\Pi x \in A)(\Pi y \in A)(x<y \to y<z \to x<z)$$

and

$$Q(A,<) = (\Pi f \in N \to A)(\,(\Pi n \in N)(f(n+1)<f(n)) \to \bot\,),$$

which express that $<$ is transitive and free from infinite descending chains, both hold. Note that an ordering without descending chains is necessarily irreflexive,

because, if $a<a$, then $\ldots<a<\ldots<a<a$ is an infinite descending chain and we get a contradiction.

Remembering the representation of propositions as types and the interpretation of the notion of such that,

$$U=(\Sigma A\in V)(\Sigma<\in A\to A\to V)(P(A,<)\times Q(A,<))$$

is the type of all orderings without infinite descending chains. On U we define a binary relation $<_U$ by putting

$$(A,<_A,p_A,q_A)<_U(B,<_B,p_B,q_B)=(\Sigma f\in A\to B)(\Sigma b\in B)$$
$$((\Pi x\in A)(\Pi y\in A)(x<_A y\to f(x)<_B f(y))\ \times\ (\Pi x\in A)(f(x)<_B b)),$$

that is, one ordering of the kind that we are considering is defined to be less than another if there exists an order preserving map from the first to the second and an element of the second ordering which dominates the range of this map.

The ordering $<_U$ is transitive. Suppose namely that

$$(A,<_A,p_A,q_A)<_U(B,<_B,p_B,q_B)<_U(C,<_C,p_C,q_C),$$

that is, that there are order preserving maps $f\in A\to B$ and $g\in B\to C$ and elements $b\in B$ and $c\in C$ that dominate their respective ranges. Then the composition of f and g is an order preserving map from A to C whose range is dominated by c so that

$$(A,<_A,p_A,q_A)<_U(C,<_C,p_C,q_C).$$

We have now constructed a proof $p_U\in P(U,<_U)$.

The ordering $<_U$ has no infinite descending chains. Suppose namely that

$$(A_{n+1},<_{n+1},p_{n+1},q_{n+1})<_U(A_n,<_n,p_n,q_n),\ \ n=0,1,\ldots,$$

and let f_n be the order preserving function that maps A_{n+1} into A_n and a_n the object of type A_n that dominates its range. Then

$$\ldots<_0 f_0(f_1(\ldots f_n(a_{n+1})\ldots))<_0\ldots<_0 f_0(f_1(a_2))<_0 f_0(a_1)<_0 a_0$$

so that we get an infinite descending chain in A_0 contrary to the assumption that $<_0$ is an ordering without such chains. We have now constructed a proof $q_U\in Q(U,<_U)$.

From $U\in V$, $<_U\in U\to U\to V$, $p_U\in P(U,<_U)$ and $q_U\in Q(U,<_U)$ we can conclude

$$(U,<_U,p_U,q_U)\in U.$$

The next step is to show that this is a maximal element of U with respect to the ordering $<_U$, that is, that

$$(A,<,p,q)<_U(U,<_U,p_U,q_U)$$

for all

$$(A, <, p, q) \in U.$$

To this end, we let f be the function which takes an object a of type A into the segment of A determined by a, that is,

$$f(a) = (A_a, <_a, p_a, q_a)$$

where, remembering the interpretation of such that,

$$A_a = (\Sigma x \in A)(x < a),$$

$<_a$ is the restriction of $<$ to A and p_a and q_a are the obvious proofs that $<_a$ is transitive and free from infinite descending chains. We have to show that f is order preserving and that its range has a dominating element. Suppose $a < b$. Then

$$f(a) <_U f(b)$$

because the injection of A_a in to A_b is order preserving and its range is dominated by the pair of type A_b which consists of a and the proof of $a < b$. As the element of type U which is to dominate the range of f we can take $(A, <, p, q)$ itself. Indeed, if a is an arbitrary object of type A, then

$$f(a) = (A_a, <_a, p_a, q_a) <_U (A, <, p, q)$$

because the injection of A_a into A is order preserving and its range is dominated by a.

We have now shown that $(U, <_U, p_U, q_U)$ is a maximal element of U with respect to the ordering $<_U$. But $(U, <_U, p_U, q_U)$ is itself an object of type U and hence

$$(U, <_U, p_U, q_U) <_U (U, <_U, p_U, q_U).$$

This, however, is impossible, because we have shown $<_U$ to be an ordering without infinite descending chains and, as remarked above, such an ordering is necessarily irreflexive.

2. FORMALIZATION OF AN INTUITIONISTIC THEORY OF TYPES.

2.1. Notational conventions. There is no need to give a list of all the formal symbols. Suffice it to say that it should contain all the symbols that are used in the following except the square brackets which will be reserved for the substitution operation. Thus, $b[x]$ denotes an expression in which there may be some free occurrences of the variable x, and $b[a]$ denotes the result of substituting the expression a for all free occurrences of the variable x in $b[x]$. Free and bound variables are defined by stipulating that all free occurrences of x in $b[x]$ become bound in $(\lambda x)b[x]$ and, similarly, that all free occurrences of x in $B[x]$ become bound in $(\Pi x \in A)B[x]$ and $(\Sigma x \in A)B[x]$.

By a simultaneous induction, we shall generate certain symbolic expressions called *types* and, for every type, certain other expressions called the *terms* of that type. The rules of type and term formation are such that, when the formal symbols are given their abstract interpretation as described in the previous chapter, it becomes clear that a type A denotes an abstract type and that a term a of type A denotes an abstract object of the type denoted by A. I shall use the notation $a \in A$ to express that a is a term of type A.

There will be *variables* each of which has a unique type associated with it, and a term or type will always *depend* on a certain finite number of variables. The notion of dependence is defined inductively by stipulating that a term or type depends on all its free variables as well as on all variables on which the types of its free variables depend. Consequently, a term or type is closed if and only if it depends on no variables at all. There will be *variable restrictions* prohibiting us to bind a variable in a term or type if there is a free variable in the said term or type whose type depends on the variable in question. These variable restrictions contain as special cases those stated by Gentzen for his system of natural deduction for first order logic. To avoid explicit mention of the variable restrictions, it will be tacitly assumed that a term which is denoted by $b[x_1, \ldots, x_n]$ contains no free variable distinct from $x_{m+1}, \ldots, x_n$ whose type depends on x_m, $m = 1, \ldots, n$. The same notational convention will be used for types. Thus, for instance, when saying that $b[x]$ is a term of type $B[x]$, it is tacitly assumed that there is no free variable in $b[x]$ or $B[x]$ whose type depends on x.

2.2. Types.

2.2.1. If P is an n-ary *type constant* with arguments of types $A_1, \ldots, A_n[x_1, \ldots, x_{n-1}]$ and $a_1, \ldots, a_n$ are terms of types $A_1, \ldots, A_n[a_1, \ldots, a_{n-1}]$, respectively, then $P(a_1, \ldots, a_n)$ is a type. Here, for $m = 1, \ldots, n$, the variable x_m is of the type $A_m[x_1, \ldots, x_{m-1}]$ which must not depend on any other variables than the explicitly exhibited $x_1, \ldots, x_{m-1}$. The type constants correspond to the predicate constants in ordinary first order predicate logic.

2.2.2. If x is a variable of type A and $B[x]$ is a type, then $(\Pi x \in A)B[x]$ is a type. When B does not contain x free, $(\Pi x \in A)B$ is abbreviated $A \to B$.

2.2.3. If x is a variable of type A and $B[x]$ is a type, then $(\Sigma x \in A)B[x]$ is a type. When B does not contain x free, $(\Pi x \in A)B$ is abbreviated $A \times B$.

2.2.4. If A and B are types, then $A + B$ is a type.

2.2.5. V is a type.

2.2.6. A term of type V is a *small* type. This is the clause which links the generation of the types with the generation of the terms.

A type which is not small is *large*. Thus, for instance, V is a large type. A type is large if and only if it contains an ocurrence of V or a type constant P, and hence it can be mechanically decided whether a type is small or large.

2.3. Terms. Each rule of term formation will be classified à la Gentzen 1934 as an introduction or elimination rule associated with one of the basic types or type forming operations.

2.3.1. Variables. If x is a *variable* of type A, then x is a term of type A. We are not allowed to introduce a variable x of type A unless x is distinct from all the variables on which the type A depends. Also, as remarked earlier, the type of a free variable must always be uniquely associated with the variable in question. We shall not care about the naming of bound variables.

2.3.2. Constants. If a is an *object constant* of type A, then a is a term of type A. The type of an object constant must always be closed. The object constants correspond, on the one hand, to the individual constants and function symbols in ordinary first order predicate logic and, on the other hand, to the axioms of a first order theory.

2.3.3. Π-introduction or λ-abstraction. If x is a variable of type A and $b[x]$ is a term of type $B[x]$, then $(\lambda x)b[x]$ is a term of type $(\Pi x \in A)B[x]$.

2.3.4. Π-elimination or application. If a and b are terms of types A and $(\Pi x \in A)B[x]$, respectively, then $b(a)$ is a term of type $B[a]$.

2.3.5. Σ-introduction or pairing. Let x be a variable of type A and $B[x]$ a type. Then, if a and b are terms of types A and $B[a]$, respectively, (a, b) is a term of type $(\Sigma x \in A)B[x]$.

2.3.6. Σ-elimination. Let x, y and z be variables of type A, $B[x]$ and $(\Sigma x \in A)B[x]$, respectively, and let $C[z]$ be a type. Then, if c and $d[x, y]$ are terms of types $(\Sigma x \in A)B[x]$ and $C[(x, y)]$, respectively, $E(c, (\lambda x)(\lambda y)d[x, y])$ is a term of type $C[c]$.

2.3.7. $+$-introduction or injection. If a is a term of type A, then $i(a)$ is a term of type $A + B$. Similarly, if b is a term of type B, then $j(b)$ is term of type $A + B$.

2.3.8. $+$-elimination or definition by cases. Let x, y and z be variables of types A, B and $A + B$, respectively, and let $C[z]$ be a type. Then, if c, $d[x]$ and $e[y]$ are terms of types $A + B$, $C[i(x)]$ and $C[j(y)]$, respectively, $D(c, (\lambda x)d[x], (\lambda y)e[y])$ is a term of type $C[c]$.

2.3.9. N_n-introduction. $1, \ldots, n$ are terms of type N_n.

2.3.10. N_n-elimination. Let z be a variable of type N_n and $C[z]$ a type. Then, if c, c_1, $\ldots$, c_n are terms of types N_n, $C[1]$, $\ldots$, $C[n]$, respectively, $R_n(c, c_1, \ldots, c_n)$ is a term of type $C[c]$.

2.3.11. N-introduction or Peano's first and second axioms. 0 is term of type N. If a is a term of type N, so is $s(a)$. A term of type N which has the form $s(s(\ldots s(0)\ldots))$ is called a *numeral*.

2.3.12. N-elimination or recursion. Let x and y be variables of types N and $C[x]$, respectively. Then, if c, d and $e[x,y]$ are terms of types N, $C[0]$ and $C[s(x)]$, respectively, $R(c,d,(\lambda x)(\lambda y)e[x,y])$ is a term of type $C[c]$.

2.3.13. V-introduction or the reflection principle. N_0, N_1, ...and N are terms of type V. If A and B are terms of type V, then so is $A+B$. If A and $B[x]$ are terms of type V, x being a variable of type A, then $(\Pi x\in A)B[x]$ and $(\Sigma x\in A)B[x]$ are terms of type V. N_0 and N_1 are alternatively denoted by $\bot$ and $\top$, respectively.

2.3.14. Type conversion. This is a structural rule, that is, a rule which is to be considered neither as an introduction rule nor as an elimination rule. If a is a term of type A which converts to a type B, then a is a term of type B. It remains for us to define the notion of conversion. Before doing this, however, it will be convenient to represent the rules of term formation by schemata similar to those used by Gentzen 1934 in his system of natural deduction for first order predicate logic.

Π-introduction
$$\frac{\begin{array}{c}x\in A\\ b[x]\in B[x]\end{array}}{(\lambda x)b[x]\in(\Pi x\in A)B[x]}$$

Π-elimination
$$\frac{b\in(\Pi x\in A)B[x]\qquad a\in A}{b(a)\in B[a]}$$

Σ-introduction
$$\frac{a\in A\qquad b\in B[a]}{(a,b)\in(\Sigma x\in A)B[x]}$$

Σ-elimination
$$\frac{\begin{array}{cc}&x\in A\quad y\in B[x]\\ c\in(\Sigma x\in A)B[x]\quad & d[x,y]\in C[(x,y)]\end{array}}{E(c,(\lambda x)(\lambda y)d[x,y])\in C[c]}$$

$+$-introduction
$$\frac{a\in A}{i(a)\in A+B}\qquad\frac{b\in B}{j(b)\in A+B}$$

$+$-elimination
$$\frac{\begin{array}{ccc}&x\in A&y\in B\\ c\in A+B\quad & d[x]\in C[i(x)]\quad & e[y]\in C[j(y)]\end{array}}{D(c,(\lambda x)d[x],(\lambda y)e[y])\in C[c]}$$

N_n-introduction
$$1\in N_n\qquad\ldots\qquad n\in N_n$$

N_n-elimination
$$\frac{c \in N_n \quad c_1 \in C[1] \quad \dots \quad c_n \in C[n]}{R_n(c, c_1, \dots, c_n) \in C[c]}$$

N-introduction
$$0 \in N \qquad \frac{a \in N}{s(a) \in N}$$

N-elimination
$$\frac{c \in N \quad d \in C[0] \quad \begin{matrix} x \in N \quad y \in C[x] \\ e[x,y] \in C[s(x)] \end{matrix}}{R(c, d, (\lambda x)(\lambda y)e[x,y]) \in C[c]}$$

V-introduction
$$N_0 \in V \qquad N_1 \in V \qquad \dots \qquad N \in V$$

$$\frac{A \in V \quad B \in V}{A + B \in V} \qquad \frac{\begin{matrix} x \in A \\ A \in V \quad B[x] \in V \end{matrix}}{(\Pi x \in A)B[x] \in V} \qquad \frac{\begin{matrix} x \in A \\ A \in V \quad B[x] \in V \end{matrix}}{(\Sigma x \in A)B[x] \in V}$$

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

2.4. Contraction, reduction and conversion. We shall be concerned with contraction, reduction and conversion of terms as well as types. However, in order to show that the terms and types are closed under reduction, we need a certain combinatorial property, the so-called Church-Rosser property, which will be proved for a class of in general meaningless formal *expressions* which is wide enough to include both the terms and the types.

2.4.1. Definition of formal expressions.

2.4.1.1. If $a_1, \dots, a_n$ are expressions and P is an n-ary type constant, then $P(a_1, \dots, a_n)$ is an expression.

2.4.1.2. If x is a variable and A and $B[x]$ are expressions, then $(\Pi x \in A)B[x]$ and $(\Sigma x \in A)B[x]$ are expressions.

2.4.1.3. If A and B are expressions, then so is $A + B$.

2.4.1.4. N_0, N_1, $\dots$, N and V are expressions.

2.4.1.5. Variables and object constants are expressions.

2.4.1.6. If $b[x]$ is an expression, then so is $(\lambda x)b[x]$.

2.4.1.7. If a and b are expressions, then so is $b(a)$.

2.4.1.8. If a and b are expressions, then so is (a, b).

2.4.1.9. If c and $d[x, y]$ are expressions, then so is $E(c, (\lambda x)(\lambda y)d[x, y])$.

2.4.1.10. If a and b are expressions, then so are $i(a)$ and $j(b)$.

2.4.1.11. If c, $d[x]$ and $e[y]$ are expressions, then so is $D(c, (\lambda x)d[x], (\lambda y)e[y])$.

2.4.1.12. $1, \dots, n$ are expressions.

2.4.1.13. If c, $c_1, \dots, c_n$ are expressions, then so is $R_n(c, c_1, \dots, c_n)$.

2.4.1.14. 0 is an expression, and, if a is an expression, then so is $s(a)$.

2.4.1.15. If c, d and $e[x, y]$ are expressions, then so is $R(c, d, (\lambda x)(\lambda y)e[x, y])$.

2.4.2. Rules of contraction.

$$
\begin{aligned}
(\lambda x)b[x](a) &\ \text{contr}\ b[a],\\
E((a,b),(\lambda x)(\lambda y)d[x,y]) &\ \text{contr}\ d[a,b],\\
\begin{cases} D(i(a),(\lambda x)d[x],(\lambda y)e[y]) & \text{contr}\ d[a],\\ D(j(b),(\lambda x)d[x],(\lambda y)e[y]) & \text{contr}\ e[b], \end{cases}&\\
\begin{cases} R_n(1,c_1,\ldots,c_n) & \text{contr}\ c_1,\\ \qquad\vdots & \\ R_n(n,c_1,\ldots,c_n) & \text{contr}\ c_n, \end{cases}&\\
\begin{cases} R(0,d,(\lambda x)(\lambda y)e[x,y]) & \text{contr}\ d,\\ R(s(a),d,(\lambda x)(\lambda y)e[x,y]) & \text{contr}\ e[a,R(a,d,(\lambda x)(\lambda y)e[x,y])]. \end{cases}&
\end{aligned}
$$

An expression which has the form of the left hand member of one of the rules of contraction is called a *redex* and the corresponding right hand member is its *contractum*. An expression a *reduces* to an expression b, abbreviated a red b, if b can be obtained from a by repeated contractions of parts of the expression a, and an expression is *irreducible* or *normal* if it cannot be further reduced. Finally, an expression a is said to *convert* into an expression b, abbreviated a conv b, if there is an expression c such that both a red c and b red c.

2.4.3. Church-Rosser property. *If a red b and a red c, then there is an expression d such that b red d and c red d.*

The proof given below is an adaptation of a proof for the type free combinator calculus shown to me by William Tait. The idea is to introduce a suitable measure of the length of the sequence of contractions which reduces an expression a to an expression b. We shall say that a reduces in one step and write a red_1 b if b is obtained by contracting some, possibly all or none, of the redexes in a, starting from within and proceeding outwards. (Of course, even if we contract all redexes that occur in a certain expression, we do not necessarily obtain a normal one, because new redexes may arise when the old ones are contracted.) Reduction in n steps is defined inductively by putting a red_0 a and letting a red_{n+1} c mean that a red_n b and b red_1 c for some b. Clearly, a red b if and only if a red_n b for some n. (Indeed, n can be taken to be the total number of contractions that are carried out when reducing a to b.)

2.4.3.1. Lemma. *If a red_1 c and $b[x]$ red_1 $d[x]$ then $b[a]$ red_1 $d[c]$.*

This is obvious from the definition of one step reduction.

2.4.3.2. Lemma. *If a red_1 b and a red_1 c, then there is an expression d such that b red_1 d and c red_1 d.*

The proof is by induction on the construction of the expression a. All cases in which a does not have the form of a redex are handled immediately by means of the induction hypothesis. Equally trivial are the cases when a is a redex which is contracted neither in b nor in c. There remain the cases when a is a redex which is contracted either in one of b and c, say c, or in both.

2.4.3.2.1. a has the form $(\lambda x)a_2[x](a_1)$. Then b has the form $(\lambda x)b_2[x](b_1)$ or $b_2[b_1]$ and c has the form $c_2[c_1]$ where

$$a_1 \text{ red}_1\ b_1, \quad a_2[x] \text{ red}_1\ b_2[x],$$
$$a_1 \text{ red}_1\ c_1, \quad a_2[x] \text{ red}_1\ c_2[x].$$

By induction hypothesis, we can find d_1 and $d_2[x]$ such that

$$b_1 \text{ red}_1\ d_1, \quad b_2[x] \text{ red}_1\ d_2[x],$$
$$c_1 \text{ red}_1\ d_1, \quad c_2[x] \text{ red}_1\ d_2[x].$$

But then $(\lambda x)b_2[x](b_1)$ red_1 $d_2[d_1]$ by the definition of one step reduction and $b_2[b_1]$ and $c_2[c_1]$ red_1 $d_2[d_1]$ by the previous lemma so that d can be taken to be $d_2[d_1]$.

2.4.3.2.2. a has the form $E((a_1,a_2),(\lambda x)(\lambda y)a_3[x,y])$. Then b has the form $E((b_1,b_2),(\lambda x)(\lambda y)b_3[x,y])$ or $b_3[b_1,b_2]$ and c has the form $c_3[c_1,c_2]$ where

$$a_1 \text{ red}_1\ b_1, \quad a_2 \text{ red}_1\ b_2, \quad a_3[x,y] \text{ red}_1\ b_3[x,y],$$
$$a_1 \text{ red}_1\ c_1, \quad a_2 \text{ red}_1\ c_2, \quad a_3[x,y] \text{ red}_1\ c_3[x,y].$$

By induction hypothesis, we can find d_1, d_2 and $d_3[x,y]$ such that

$$b_1 \text{ red}_1\ d_1, \quad b_2 \text{ red}_1\ d_2, \quad b_3[x,y] \text{ red}_1\ d_3[x,y],$$
$$c_1 \text{ red}_1\ d_1, \quad c_2 \text{ red}_1\ d_2, \quad c_3[x,y] \text{ red}_1\ d_3[x,y].$$

But then $E((b_1,b_2),(\lambda x)(\lambda y)b_3[x,y])$ red_1 $d_3[d_1,d_2]$ by the definition of one step reduction and $b_3[b_1,b_2]$ and $c_3[c_1,c_2]$ red_1 $d_3[d_1,d_2]$ by the previous lemma so that d can be taken to be $d_3[d_1,d_2]$.

2.4.3.2.3. a has the form $D(i(a_1),(\lambda x)a_2[x],(\lambda y)a_3[y])$. Then b has the form $D(i(b_1),(\lambda x)b_2[x],(\lambda y)b_3[y])$ or $b_2[b_1]$ and c has the form $c_2[c_1]$ where

$$a_1 \text{ red}_1\ b_1, \quad a_2[x] \text{ red}_1\ b_2[x], \quad a_3[y] \text{ red}_1\ b_3[y],$$
$$a_1 \text{ red}_1\ c_1, \quad a_2[x] \text{ red}_1\ c_2[x].$$

By induction hypothesis, we can find d_1 and $d_2[x]$ such that

$$b_1 \text{ red}_1\ d_1, \quad b_2[x] \text{ red}_1\ d_2[x],$$
$$c_1 \text{ red}_1\ d_1, \quad c_2[x] \text{ red}_1\ d_2[x].$$

But then $D(i(b_1),(\lambda x)b_2[x],(\lambda y)b_3[y])$ red_1 $d_2[d_1]$ by the definition of one step reduction and $b_2[b_1]$ and $c_2[c_1]$ red_1 $d_2[d_1]$ by the previous lemma so that d can be taken to be $d_2[d_1]$.

2.4.3.2.4. a has the form $D(j(a_1),(\lambda x)a_2[x],(\lambda y)a_3[y])$. This case is completely symmetric to the previous one.

2.4.3.2.5. a has the form $R_n(m, a_1, \ldots, a_n)$. Then b has the form $R_n(m, b_1, \ldots, b_n)$ or b_m and c has the form c_m where

$$a_m \text{ red}_1 \ b_m,$$
$$a_m \text{ red}_1 \ c_m.$$

By induction hypothesis, we can find d_m such that

$$b_m \text{ red}_1 \ d_m,$$
$$c_m \text{ red}_1 \ d_m.$$

But then $R_n(m, b_1, \ldots, b_n)$ red_1 d_m by the definition of one step reduction and b_m and c_m red_1 d_m so that d can be taken to be d_m.

2.4.3.2.6. a has the form $R(0, a_2, (\lambda x)(\lambda y)a_3[x, y])$. Then b has the form $R(0, b_2, (\lambda x)(\lambda y)b_3[x, y])$ or b_2 and c has the form c_2 where

$$a_2 \text{ red}_1 \ b_2, \quad a_3[x, y] \text{ red}_1 \ b_3[x, y],$$
$$a_2 \text{ red}_1 \ c_2.$$

By induction hypothesis, we can find d_2 such that

$$b_2 \text{ red}_1 \ d_2,$$
$$c_2 \text{ red}_1 \ d_2.$$

But then $R(0, b_2, (\lambda x)(\lambda y)b_3[x, y])$ red_1 d_2 by the definition of one step reduction and b_2 and c_2 red_1 d_2 so that d can be taken to be d_2.

2.4.3.2.7. a has the form $R(s(a_1), a_2, (\lambda x)(\lambda y)a_3[x, y])$. Then b has the form $R(s(b_1), b_2, (\lambda x)(\lambda y)b_3[x, y])$ or $b_3[b_1, R(b_1, b_2, (\lambda x)(\lambda y)b_3[x, y])]$ and c has the form $c_3[c_1, R(c_1, c_2, (\lambda x)(\lambda y)c_3[x, y])]$ where

$$a_1 \text{ red}_1 \ b_1, \quad a_2 \text{ red}_1 \ b_2, \quad a_3[x, y] \text{ red}_1 \ b_3[x, y],$$
$$a_1 \text{ red}_1 \ c_1, \quad a_2 \text{ red}_1 \ c_2, \quad a_3[x, y] \text{ red}_1 \ c_3[x, y].$$

By induction hypothesis, we can find d_1, d_2 and $d_3[x, y]$ such that

$$b_1 \text{ red}_1 \ d_1, \quad b_2 \text{ red}_1 \ d_2, \quad b_3[x, y] \text{ red}_1 \ d_3[x, y],$$
$$c_1 \text{ red}_1 \ d_1, \quad c_2 \text{ red}_1 \ d_2, \quad c_3[x, y] \text{ red}_1 \ d_3[x, y].$$

Let d be the expression $d_3[d_1, R(d_1, d_2, (\lambda x)(\lambda y)d_3[x, y])]$. Then $R(s(b_1), b_2, (\lambda x)(\lambda y)b_3[x, y])$ red_1 d and

$$R(b_1, b_2, (\lambda x)(\lambda y)b_3[x, y]) \text{ red}_1 \ R(d_1, d_2, (\lambda x)(\lambda y)d_3[x, y])$$

and

$$R(c_1, c_2, (\lambda x)(\lambda y)c_3[x, y]) \text{ red}_1 \ R(d_1, d_2, (\lambda x)(\lambda y)d_3[x, y])$$

by the definition of one step reduction so that $b_3[b_1, R(b_1, b_2, (\lambda x)(\lambda y)b_3[x, y])]$ and $c_3[c_1, R(c_1, c_2, (\lambda x)(\lambda y)c_3[x, y])]$ red_1 d by the previous lemma as desired.

2.4.3.3. Lemma. *If a red_m b and a red_n c then there is an expression d such that b red_n d and c red_m d.*

This folllows by mn applications of the previous lemma. The proof of the Church-Rosser property is now complete.

2.4.4. Corollary. *The relation a conv b is an equivalence relation.*

The reflexivity and symmetry are both obvious from the definition. To prove the transitivity, suppose that a conv b and b conv c. By the definition of the convertibility relation, this means that there are expressions d and e such that a red d, b red d, b red e and c red e. Because of the Church-Rosser property, we can find an expression f such that d red f and e red f. Since the reducibility relation is transitive, a red f and c red f so that a conv c as desired.

It is a consequence of the transitivity of the convertibility relation that a sequence of successive applications of the rule of type conversion can be condensed into one application of the same rule. Thus, whenever convenient, we can assume that there is at most one (or even precisely one) application of the rule of type conversion between two successive applications of the non structural rules of term formation.

2.4.5. Uniqueness of normal forms. *An expression can convert into at most one normal expression.*

First note that, because of the Church-Rosser property, an expression which converts into a normal expression must necessarily reduce to it. So suppose that a red b and a red c where a is an arbitrary expression and b and c are both normal. By the Church-Rosser property, there is an expression d such that b red d and c red d. Since b and c are both normal, they must be equal to d and hence equal to each other. Remember that equality means syntactical equality neglecting differences in the naming of bound variables.

2.4.6. Definitional equality. Two types A and B are said to be *definitionally equal* provided A conv B. Also a term a of type A is definitionally equal to a term b of type B if both a conv b and A conv B. Note that, because of the rule of type conversion, two terms are of definitionally equal types if and only if they are of the same type. Two definitionally equal types denote the same abstract type, and, similarly, two definitionally equal terms denote the same object of the abstract type denoted by their types. Thus, definitional equality is a relation between linguistic expressions and not between the abstract entities which they denote (and which are the same). This explains why the rule of type conversion, unlike all the other rules of term formation, has no counterpart on the informal level. (It is superfluous to say that if a is an object of type A and the types A and B are the same, then a is an object of type B.)

Because of the representation of propositions as types and hence of proofs as mathematical objects, the relation of definitional equality just introduced embraces at the same time Tait's 1967 notion of definitional equality between the terms of Gödel's 1958 theory of primitive recursive functionals of finite type

and the notion of definitional equality between derivations representing proofs that I have proposed earlier (see Prawitz 1971).

2.4.7. Theorem. *The terms of a given type are closed under reduction.*

This means that, if a term of a certain type reduces to an expression, then this expression is actually a term of the same type. Clearly, it suffices to prove this when the expression is obtained from the given term by one step reduction. The proof is by induction on the derivation of the given term. Several cases have to be distinguished.

2.4.7.1. The derivation of the given term has the form

$$\dfrac{\begin{array}{c}x \in A\\ \vdots\\ b[x] \in B[x]\end{array}}{(\lambda x)b[x] \in (\Pi x \in A)B[x]}$$

and $(\lambda x)b[x]$ red$_1$ $(\lambda x)d[x]$ because $b[x]$ red$_1$ $d[x]$. By induction hypothesis, $d[x] \in B[x]$ under the assumption $x \in A$, and hence $(\lambda x)d[x] \in (\Pi x \in A)B[x]$ follows by Π-introduction.

2.4.7.2. The derivation of the given term has the form

$$\dfrac{\begin{array}{c}\vdots\\ b \in (\Pi x \in A)B[x]\end{array} \qquad \begin{array}{c}\vdots\\ a \in A\end{array}}{b(a) \in B[a]}$$

and $b(a)$ red$_1$ $d(c)$ because a red$_1$ c and b red$_1$ d. By induction hypothesis, $c \in A$ and $d \in (\Pi x \in A)B[x]$, from which $d(c) \in B[c]$ follows by Π-elimination. Type conversion then yields $d[c] \in B[a]$ as desired.

2.4.7.3. The derivation of the given term has the form

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}x \in C\\ \vdots\\ b[x] \in D[x]\end{array}}{(\lambda x)b[x] \in (\Pi x \in C)D[x]} \qquad \vdots}{(\lambda x)b[x] \in (\Pi x \in A)B[x]} \qquad a \in A}{(\lambda x)b[x](a) \in B[a]}$$

where $(\Pi x \in A)B[x]$ conv $(\Pi x \in C)D[x]$, that is, A conv B and $B[x]$ conv $D[x]$, and $(\lambda x)b[x](a)$ red$_1$ $d[c]$ because a red$_1$ c and $b[x]$ red$_1$ $d[x]$. By induction

hypothesis, $c \in A$ and $d[x] \in D[x]$ under the assumption $x \in C$. The derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{c \in A}}{c \in C}\ \vdots}{\dfrac{d[c] \in D[c]}{d[c] \in B[a]}}$$

shows that $d[c]$ is indeed a term of type $B[a]$.

2.4.7.4. The derivation of the given term has the form

$$\dfrac{\dfrac{\vdots}{a \in A} \qquad \dfrac{\vdots}{b \in B[a]}}{(a,b) \in (\Sigma x \in A)B[x]}$$

and (a,b) red_1 (c,d) because a red_1 c and b red_1 d. By induction hypothesis, $c \in A$ and $d \in B[a]$. Type conversion yields $d \in B[c]$, and $(c,d) \in (\Sigma x \in A)B[x]$ then follows by Σ-introduction.

2.4.7.5. The derivation of the given term has the form

$$\dfrac{\dfrac{\vdots}{c \in (\Sigma x \in A)B[x]} \qquad \dfrac{\begin{array}{c} x \in A \quad y \in B[x] \\ \ddots \quad \iddots \\ d[x,y] \in C[(x,y)] \end{array}}{E(c, (\lambda x)(\lambda y)d[x,y]) \in C[c]}$$

and $E(c, (\lambda x)(\lambda y)d[x,y])$ red_1 $E(f, (\lambda x)(\lambda y)g[x,y])$ because c red_1 f and $d[x,y]$ red_1 $g[x,y]$. By induction hypothesis, $f \in (\Sigma x \in A)B[x]$ and $g[x,y] \in C[(x,y)]$ under the assumptions $x \in A$ and $y \in B[x]$, and hence $E(f, (\lambda x)(\lambda y)g[x,y]) \in C[f]$ follows by Σ-elimination. Type conversion then yields $E(f, (\lambda x)(\lambda y)g[x,y]) \in C[c]$ as desired.

2.4.7.6. The derivation of the given term has the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{a \in C} \qquad \dfrac{\vdots}{b \in D[a]}}{(a,b) \in (\Sigma x \in C)D[x]} \qquad \dfrac{\begin{array}{c} x \in A \quad y \in B[x] \\ \ddots \quad \iddots \\ d[x,y] \in C[(x,y)] \end{array}}{(a,b) \in (\Sigma x \in A)B[x]}}{E((a,b), (\lambda x)(\lambda y)d[x,y]) \in C[(a,b)]}$$

where $(\Sigma x \in A)B[x]$ conv $(\Sigma x \in C)D[x]$, that is, A conv C and $B[x]$ conv $D[x]$, and $E((a,b), (\lambda x)(\lambda y)d[x,y])$ red_1 $g[c,d]$ because a red_1 c, b red_1 d and $d[x,y]$ red_1 $g[x,y]$. By induction hypothesis, $c \in C$, $d \in D[a]$ and $g[x,y] \in C[(x,y)]$ under the assumptions $x \in A$ and $y \in B[x]$. The derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{c \in C}}{c \in A} \quad \dfrac{\dfrac{\vdots}{d \in D[a]}}{d \in B[c]}}{\dfrac{g[c,d] \in C[(c,d)]}{g[c,d] \in C[(a,b)]}}$$

shows that $g[c,d]$ indeed is a term of type $C[(a,b)]$.

2.4.7.7. The derivation of the given term has the form

$$\dfrac{\dfrac{\vdots}{a \in A}}{i(a) \in A+B}$$

and $i(a)$ red_1 $i(c)$ because a red_1 c. By induction hypothesis, we get $c \in A$, from which $i(a) \in A+B$ follows by $+$-introduction. The other rule of $+$-introduction is treated in the same way.

2.4.7.8. The derivation of the given term has the form

$$\dfrac{\dfrac{\vdots}{c \in A+B} \quad \dfrac{\dfrac{x \in A}{\vdots}}{d[x] \in C[i(x)]} \quad \dfrac{\dfrac{y \in B}{\vdots}}{e[y] \in C[j(y)]}}{D(c,(\lambda x)d[x],(\lambda y)e[y]) \in C[c]}$$

and $D(c,(\lambda x)d[x],(\lambda y)e[y])$ red_1 $D(f,(\lambda x)g[x],(\lambda y)h[y])$ because c red_1 f, $d[x]$ red_1 $g[x]$ and $e[y]$ red_1 $h[y]$. By induction hypothesis, $f \in A+B$ and $g[x] \in C[i(x)]$ and $h[y] \in C[j(y)]$ under the assumptions $x \in A$ and $y \in B$, respectively, and hence $D(f,(\lambda x)g[x],(\lambda y)h[y]) \in C[f]$ follows by $+$-elimination. Type conversion then yields $D(f,(\lambda x)g[x],(\lambda y)h[y]) \in C[c]$ as desired.

2.4.7.9. The derivation of the given term has the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{a \in C}}{i(a) \in C+D}}{i(a) \in A+B} \quad \dfrac{\dfrac{x \in A}{\vdots}}{d[x] \in C[i(x)]} \quad \dfrac{\dfrac{y \in B}{\vdots}}{e[y] \in C[j(y)]}}{D(i(a),(\lambda x)d[x],(\lambda y)e[y]) \in C[i(a)]}$$

where $A+B$ conv $C+D$, that is, A conv C and B conv D, and $D(i(a),(\lambda x)d[x],(\lambda y)e[y])$ red_1 $g[c]$ because a red_1 c and $d[x]$ red_1 $g[x]$. By induction hypothesis,

$c \in C$ and $g[x] \in C[i(x)]$ under the assumption $x \in A$. The derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{c \in C}}{c \in A}\ \vdots}{\dfrac{g[c] \in C[i(c)]}{g[c] \in C[i(a)]}}$$

shows that $g[c]$ is indeed a term of type $C[i(a)]$. The case when $i(a)$ is replaced by $j(b)$ is treated in the same way.

2.4.7.10. The terms $1, \ldots, n$ of type N_n cannot reduce to anything but themselves, and hence this case is trivial.

2.4.7.11. The derivation of the given term has the form

$$\dfrac{\begin{array}{cccc}\vdots & \vdots & & \vdots \\ c \in N_n & c_1 \in C[1] & \ldots & c_n \in C[n]\end{array}}{R_n(c, c_1, \ldots, c_n) \in C[c]}$$

and $R_n(c, c_1, \ldots, c_n)$ red_1 $R_n(f, f_1, \ldots, f_n)$ because c red_1 f, c_1 red_1 f_1, $\ldots$, c_n red_1 f_n. By induction hypothesis, we get $f \in N_n$, $f_1 \in C[1]$, $\ldots$, $f_n \in C[n]$, from which $R_n(f, f_1, \ldots, f_n) \in C[f]$ follows by N_n-elimination. Type conversion then yields $R_n(f, f_1, \ldots, f_n) \in C[c]$ as desired.

2.4.7.12. The derivation of the given term has the form

$$\dfrac{\begin{array}{cccc} & \vdots & & \vdots \\ m \in N_n & c_1 \in C[1] & \ldots & c_n \in C[n]\end{array}}{R_n(m, c_1, \ldots, c_n) \in C[c]}$$

and $R_n(m, c_1, \ldots, c_n)$ red_1 f_m because c_m red_1 f_m. By induction hypothesis, we get $f_m \in C[m]$ as desired.

2.4.7.13. The case when the given term is 0 of type N is trivial, so suppose the derivation of the given term has the form

$$\dfrac{\dfrac{\vdots}{a \in N}}{s(a) \in N}$$

and that $s(a)$ red_1 $s(c)$ because a red_1 c. By induction hypothesis, $c \in N$ and hence $s(c) \in N$ follows by N-introduction.

2.4.7.14. The derivation of the given term has the form

$$\dfrac{\begin{array}{cccc} & & x \in N & y \in C[x] \\ \vdots & \vdots & \ddots & \iddots \\ c \in N & d \in C[0] & e[x,y] \in C[s(x)] & \end{array}}{R(c, d, (\lambda x)(\lambda y)e[x,y]) \in C[c]}$$

and $R(c,d,(\lambda x)(\lambda y)e[x,y])$ red_1 $R(f,g,(\lambda x)(\lambda y)h[x,y])$ because c red_1 f, d red_1 g and $e[x,y]$ red_1 $h[x,y]$. By induction hypothesis, $f \in N$, $g \in C[0]$ and $h[x,y] \in C[s(x)]$ under the assumptions $x \in N$ and $y \in C[x]$, and hence $R(f,g,(\lambda x)(\lambda y)h[x,y]) \in C[f]$ follows by N-elimination. Type conversion then yields $R(f,g,(\lambda x)(\lambda y)h[x,y]) \in C[c]$ as desired.

2.4.7.15. The derivation of the given term has the form

$$\dfrac{0 \in N \quad d \in C[0] \quad \begin{array}{c} x \in N \quad y \in C[x] \\ \vdots \\ e[x,y] \in C[s(x)] \end{array}}{R(0,d,(\lambda x)(\lambda y)e[x,y]) \in C[0]}$$

and $R(c,d,(\lambda x)(\lambda y)e[x,y])$ red_1 g because d red_1 g. By induction hypothesis, we get $g \in C[0]$ as desired.

We also have to consider the case when the derivation of the given term has the form

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ a \in N\end{array}}{s(a) \in N} \quad \begin{array}{c}\vdots \\ d \in C[0]\end{array} \quad \begin{array}{c} x \in N \quad y \in C[x] \\ \vdots \\ e[x,y] \in C[s(x)] \end{array}}{R(s(a),d,(\lambda x)(\lambda y)e[x,y]) \in C[s(a)]}$$

and $R(s(a),d,(\lambda x)(\lambda y)e[x,y])$ red_1 $h[c,R(c,g,(\lambda x)(\lambda y)h[x,y])]$ because a red_1 c, d red_1 g and $e[x,y]$ red_1 $h[x,y]$. By induction hypothesis, $c \in N$, $g \in C[0]$ and $h[x,y] \in C[s(x)]$ under the assumptions $x \in N$ and $y \in C[x]$. The derivation

$$\dfrac{\begin{array}{c}\vdots \\ c \in N\end{array} \quad \dfrac{\begin{array}{c}\vdots \\ c \in N\end{array} \quad \begin{array}{c}\vdots \\ g \in C[0]\end{array} \quad \begin{array}{c} x \in N \quad y \in C[x] \\ \vdots \\ h[x,y] \in C[s(x)] \end{array}}{R(c,g,(\lambda x)(\lambda y)h[x,y]) \in C[c]}}{h[c,R(c,g,(\lambda x)(\lambda y)h[x,y])] \in C[s(c)]}$$

$$h[c,R(c,g,(\lambda x)(\lambda y)h[x,y])] \in C[s(a)]$$

shows that $h[c,R(c,g,(\lambda x)(\lambda y)h[x,y])]$ is indeed a term of type $C[s(a)]$.

2.4.7.16. The derivation of the given term has the form

$$\dfrac{\begin{array}{c}\vdots \\ A \in V\end{array} \quad \begin{array}{c} x \in A \\ \vdots \\ B[x] \in V \end{array}}{(\Pi x \in A)B[x] \in V}$$

and $(\Pi x \in A)B[x]$ red_1 $(\Pi x \in C)D[x]$ because A red_1 C and $B[x]$ red_1 $D[x]$. By induction hypothesis, $C \in V$ and $D[x] \in V$ under the assumption $x \in A$. The

derivation

$$\dfrac{\begin{array}{cc} & \dfrac{x \in C}{x \in A} \\ \vdots & \vdots \\ C \in V & D[x] \in C \end{array}}{(\Pi x \in C) D[x] \in V}$$

shows that $(\Pi x \in C) D[x]$ is indeed a term of type V.

The case when Σ takes the place of Π is treated in the same way, so suppose instead that the derivation of the given term has the form

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ A \in V & B \in V \end{array}}{A + B \in V}$$

and that $A + B$ red$_1$ $C + D$ because A red$_1$ C and B red$_1$ D. By induction hypothesis, $C \in V$ and $D \in V$ do that $C + D \in V$ as desired.

The case when the given term is one of N_0, N_1, ..., N is trivial because they do not reduce to anything but themselves. The proof of theorem 2.4.7 is now complete.

2.4.8. Corollary. *The types are closed under reduction.*

Suppose that a type A reduces to an expression B. We have to show that B is a type as well and use induction on the construction of A. The induction step is trivial, and so is the case when A is V. If A is a term of type V, then, by the previous theorem, B is a term of type V and hence a type. Finally, suppose that A has the form $P(a_1, \ldots, a_n)$ where a_1, ..., a_n are terms of types A_1, ..., $A_n[a_1, \ldots, a_{n-1}]$, respectively. Then B must have the form $P(b_1, \ldots, b_n)$ where a_1 red b_1, ..., a_n red b_n. By the previous theorem, b_1, ..., b_n are terms of types A_1, ..., $A_n[a_1, \ldots, a_{n-1}]$, respectively. But $A_m[a_1, \ldots, a_{m-1}]$ conv $A_m[b_1, \ldots, b_{m-1}]$ and hence, by the rule of type conversion, b_m is a term of type $A_m[b_1, \ldots, b_{m-1}]$ for $m = 1, \ldots, n$, so that $P(b_1, \ldots, b_n)$ is indeed a type.

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

$$\begin{aligned} &(\Pi x \in A)(\Sigma y \in B[x]) C[x, y] \ \rightarrow \\ &\qquad (\Sigma f \in (\Pi x \in A) B[x])\ (\Pi x \in A) C[x, f(x)]. \end{aligned}$$

To begin with, note that, if we let $p[z]$ and $q[z]$ denote the terms $E(z, (\lambda x)(\lambda y)\, x)$ and$E(z, (\lambda x)(\lambda y) y)$ which satisfy

$$\begin{cases} p[(a, b)] & \text{contr} \quad a, \\ q[(a, b)] & \text{contr} \quad b, \end{cases}$$

then

$$\frac{c \in (\Sigma x \in A)B[x]}{p[c] \in A} \qquad \frac{c \in (\Sigma x \in A)B[x]}{q[c] \in B[p[c]]}$$

hold as derived rules of term formation since they are instances of the Σ-elimination rule.

Now, suppose

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

Π-elimination gives

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

from which the derived rules of inference just stated allow us to conclude

$$p[z(x)] \in B[x] \quad \text{and} \quad q[z(x)] \in C[x,p[z(x)]].$$

Type conversion on the latter yields

$$q[z(x)] \in C[x,(\lambda x)p[z(x)](x)],$$

and we can then use Π-introduction to get

$$(\lambda x)p[z(x)] \in (\Pi x \in A)B[x]$$

as well as

$$(\lambda x)q[z(x)] \in (\Pi x \in A)C[x,(\lambda x)p[z(x)](x)].$$

Applying Σ-introduction to the last two terms, we get

$$((\lambda x)p[z(x)],(\lambda x)q[z(x)]) \in (\Sigma f \in (\Pi x \in A)B[x])(\Pi x \in A)C[x,f(x)],$$

and a final Π-introduction then shows that

$$(\lambda z)((\lambda x)p[z(x)],(\lambda x)q[z(x)])$$

is a (closed) term of type

$$\begin{aligned}&(\Pi x \in A)(\Sigma y \in B[x])C[x,y] \;\rightarrow \\ &\qquad (\Sigma f \in (\Pi x \in A)B[x])(\Pi x \in A)C[x,f(x)]\end{aligned}$$

as desired.

3. REDUCTION OF SOME OTHER FORMAL THEORIES TO THE THEORY OF TYPES.

3.1. Intuitionistic first order predicate logic.

3.1.1. Formulae are built up as usual from individual variables, function constants and predicate constants by means of the logical operators $\bot$, $\supset$, $\&$, $\vee$, $\forall$ and $\exists$. The negation $\sim A$ of a formula A is defined as $A \supset \bot$. We take the rules of inference from Gentzen 1934.

$\supset$-introduction
$$\begin{array}{c} A \\ B \\ \hline A \supset B \end{array}$$

$\supset$-elimination
$$\frac{A \supset B \quad A}{B}$$

$\&$-introduction
$$\frac{A \quad B}{A \,\&\, B}$$

$\&$-elimination
$$\frac{A \,\&\, B}{A} \qquad \frac{A \,\&\, B}{B}$$

$\vee$-introduction
$$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B}$$

$\vee$-elimination
$$\begin{array}{ccc} & A & B \\ A \vee B & C & C \\ \hline & C & \end{array}$$

$\forall$-introduction
$$\frac{B[x]}{\forall x B[x]}$$

$\forall$-elimination
$$\frac{\forall x B[x]}{B[a]}$$

$\exists$-introduction
$$\frac{B[a]}{\exists x B[x]}$$

$\exists$-elimination
$$\begin{array}{cc} & B[x] \\ \exists x B[x] & C \\ \hline & C \end{array}$$

$\bot$-elimination $\dfrac{\bot}{C}$

3.1.2. To translate this system into the theory of types, we first introduce a 0-ary type constant I^* for the type of individuals and then proceed as follows.

3.1.2.1. Translation of the language.

3.1.2.1.1. An individual variable x is mapped into a variable x^* of type I^*.

3.1.2.1.2. An n-ary function symbol f is mapped into a constant f^* of type

$$\underbrace{I^* \to \ldots \to I^*}_{n} \to I^*$$

and, if $a_1, \ldots, a_n$ are individual terms, we let $f(a_1, \ldots, a_n)^*$ be $f^*(a_1^*, \ldots, a_n^*)$.

3.1.2.1.3. An n-ary predicate constant P is mapped into an n-ary type constant P^* with all arguments of type I^*.

3.1.2.1.4. An atomic formula $P(a_1, \ldots, a_n)$ is mapped into the type $P^*(a_1^*, \ldots, a_n^*)$, and the formulae $\bot$, $A \supset B$, $A \& B$, $A \vee B$, $\forall x B[x]$ and $\exists x B[x]$ are translated into the types N_0, $A^* \to B^*$, $A^* \times B^*$, $A^* + B^*$, $(\Pi x^* \in I^*) B^*[x^*]$ and $(\Sigma x^* \in I^*) B^*[x^*]$, respectively. Note that, for every formula A, the type A^* is normal.

3.1.2.2. Translation of the derivations. By induction on the length of a derivation a of a formula A in first order logic, I shall construct a term a^* of type A^* in the theory of types.

3.1.2.2.1. Corresponding to an assumption A in first order logic, we introduce a variable x^* of type A^* in the theory of types.

3.1.2.2.2. $\supset$-introduction.

$$\dfrac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{A \supset B}$$

By induction hypothesis, we have constructed a term $b^*[x^*]$ of type A^* where x^* is the variable of type A^* that corresponds to the assumption A. The translation of the derivation of $A \supset B$ is defined to be $(\lambda x^*) b^*[x^*]$. By the rule of Π-introduction, this is a term of the type $A^* \to B^*$ which is $(A \supset B)^*$.

3.1.2.2.3. $\supset$-elimination.

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ A \supset B & A \end{array}}{B}$$

By induction hypothesis, we have constructed terms a^* and b^* of types A^* and $(A \supset B)^*$, respectively. The translation of the derivation of B is defined to be $b^*(a^*)$ which, by the rule of Π-elimination and the definition of $(A \supset B)^*$ as $A^* \to B^*$, is a term of type B^*.

3.1.2.2.4. &-introduction.

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A\,\&B}$$

By induction hypothesis, we have constructed terms a^* and b^* of types A^* and B^*, respectively. The translation of the derivation of $A\,\&B$ is defined to be (a^*, b^*). By the rule of Σ-introduction, this is a term of the type $A^* \times B^*$ which is $(A\,\&B)^*$.

3.1.2.2.5. &-elimination.

$$\dfrac{\begin{array}{c}\vdots \\ A\,\&B\end{array}}{A}$$

By induction hypothesis, we have constructed a term c^* of type $(A\,\&B)^*$. The translation of the derivation of A is defined to be $p[c^*]$, that is, $E(c^*, (\lambda x)(\lambda y)x)$, which, by the rule of Σ-elimination and the definition of $(A\,\&B)^*$ as $A^* \times B^*$ is a term of type A^*. The case when B rather than A is inferred from $A\,\&B$ is treated in the same way.

3.1.2.2.6. $\vee$-introduction.

$$\dfrac{\begin{array}{c}\vdots \\ A\end{array}}{A \vee B}$$

By induction hypothesis, we have constructed a term a^* of type A^*. The translation of the derivation of $A \vee B$ is defined to be $i(a^*)$. By the rule of $+$-introduction, this is a term of the type $A^* + B^*$ which is $(A \vee B)^*$. The case in which $A \vee B$ is inferred from B instead of A is treated in the same way.

3.1.2.2.7. $\vee$-elimination.

$$\dfrac{\begin{array}{ccc} & A & B \\ \vdots & \vdots & \vdots \\ A \vee B & C & C\end{array}}{C}$$

By induction hypothesis, we have constructed terms c^*, $d^*[x^*]$ and $e^*[y^*]$ of types $(A \vee B)^*$, C^* and C^*, respectively, where x^* and y^* are the variables corresponding to the assumptions A and B. The translation of the derivation of C is defined to be $D(c^*, (\lambda x^*)d^*[x^*], (\lambda y^*)e^*[y^*])$ which, by the rule of $+$-elimination and the definition of $(A \vee B)^*$ as $A^* + B^*$, is a term of type C^* as required.

3.1.2.2.8. $\forall$-introduction.

$$\dfrac{\begin{array}{c}\vdots \\ B[x]\end{array}}{\forall x B[x]}$$

By induction hypothesis, we have constructed a term $b^*[x^*]$ of type $B^*[x^*]$ where x^* is the variable of type I^* which correspond to the individual variable x. The translation of the derivation of $\forall x B[x]$ is defined to be $(\lambda x^*)b^*[x^*]$. By the rule of Π-introduction, this is a term of the type $(\Pi x^* \in I^*)B^*[x^*]$ which is $(\forall x B[x])^*$.

3.1.2.2.9. $\forall$-elimination.

$$\frac{\begin{array}{c}\vdots\\ \forall x B[x]\end{array}}{B[a]}$$

By induction hypothesis, we have constructed a term b^* of type $(\forall x B[x])^*$. Let the term a^* of type I^* be the translation of the individual term a. The translation of the derivation of $B[a]$ is defined to be $b^*(a^*)$ which, by the rule of Π-elimination and the definition of $(\forall x B[x])^*$ as $(\Pi x^* \in I^*)B^*[x^*]$, is a term of type $B^*[a^*]$. It only remains to remark that $B^*[a^*]$ equals $B[a]^*$.

3.1.2.2.10. $\exists$-introduction.

$$\frac{\begin{array}{c}\vdots\\ B[a]\end{array}}{\exists x B[x]}$$

By induction hypothesis, we have constructed a term b^* of type $B[a]^*$ or, what amounts to the same, $B^*[a^*]$. Let the term a^* of type I^* be the translation of the individual term a. The translation of the derivation of $\exists x B[x]$ is defined to be (a^*, b^*). By the rule of Σ-introduction, this is a term of the type $(\Sigma x^* \in I^*)B^*[x^*]$ which is $(\exists x B[x])^*$.

3.1.2.2.11. $\exists$-elimination.

$$\frac{\begin{array}{cc} & B[x]\\ \vdots & \vdots\\ \exists x B[x] & C\end{array}}{C}$$

By induction hypothesis, we have constructed terms c^* and $d^*[x^*, y^*]$ of types $(\exists x B[x])^*$ and C^*, respectively, where x^* is the translation of the individual variable x and y^* is the variable of type $B^*[x^*]$ that corresponds to the assumption $B[x]$. The translation of the derivation of C is defined to be $E(c^*, (\lambda x^*)(\lambda y^*)$ $d^*[x^*, y^*])$ which, by the rule of Σ-elimination and the definition of $(\exists x B[x])^*$ as $(\Sigma x^* \in I^*)B^*[x^*]$, is a term of type C^*.

3.1.2.2.12. $\bot$-elimination.

$$\frac{\begin{array}{c}\vdots\\ \bot\end{array}}{C}$$

By induction hypothesis, we have constructed a term c^* of type $\bot^*$. The translation of the derivation of C is defined to be $R_0(c^*)$ which, by the rule of N_0-elimination and the definition of $\bot^*$ as N_0, is a term of type C^*.

3.1.3. Consider the reduction relation between derivations in first order logic which is generated by Prawitz's 1965 rules of contraction.

$\supset$-contraction

$$\dfrac{\dfrac{\begin{array}{c}A\\\vdots\\B\end{array}}{A\supset B}\quad\begin{array}{c}\vdots\\A\end{array}}{B}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\A\\\vdots\\B\end{array}$$

&-contraction

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots\\A & B\end{array}}{A\&B}}{A}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\A\end{array}\qquad\qquad\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots\\A & B\end{array}}{A\&B}}{B}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\B\end{array}$$

$\vee$-contraction

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\A\end{array}}{A\vee B}\quad\begin{array}{c}A\\\vdots\\C\end{array}\quad\begin{array}{c}B\\\vdots\\C\end{array}}{C}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\A\\\vdots\\C\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\B\end{array}}{A\vee B}\quad\begin{array}{c}A\\\vdots\\C\end{array}\quad\begin{array}{c}B\\\vdots\\C\end{array}}{C}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\B\\\vdots\\C\end{array}$$

$\forall$-contraction

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\B[x]\end{array}}{\forall xB[x]}}{B[a]}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\B[a]\end{array}$$

$\exists$-contraction

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\B[a]\end{array}}{\exists xB[x]}\quad\begin{array}{c}B[x]\\\vdots\\C\end{array}}{C}\qquad\text{contr}\qquad\begin{array}{c}\vdots\\B[a]\\\vdots\\C\end{array}$$

The mapping of the derivations of first order logic into terms of the theory of types is an isomorphic imbedding in the sense that, if a red b, then a^* red b^*, and, conversely, if a is a derivation in first order logic and a^* red b^*, then b^* is the translation of a derivation b in first order logic such that a red b. Consequently, Prawitz's 1965 normalization theorem for first order logic is a corollary of the normalization theorem for the theory of types that will be proved later on.

3.2. Intuitionistic first order arithmetic.

3.2.1. As usual, we take the language to be the language of first order

predicate logic based on the single binary predicate constant $=$ and the four function constants, 0, $'$, $+$, and $\cdot$. We also include the propositional constant $\top$ for truth. To the rules of inference of intuitionistic first order predicate logic, we add the axiom $\top$, the induction schema

$$\frac{C[0] \qquad \begin{matrix} C[x] \\ C[x'] \end{matrix}}{C[a]}$$

and the rule of formula conversion

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

where conv is the convertibility relation which is generated by the rules of contraction

$$\begin{cases} a + 0 \text{ contr } a, \\ a + b' \text{ contr } (a + b)', \end{cases} \qquad \begin{cases} a \cdot 0 \text{ contr } 0, \\ a \cdot b' \text{ contr } a \cdot b + a, \end{cases}$$

$$\begin{cases} 0 = 0 \text{ contr } \top, \\ a' = 0 \text{ contr } \bot, \\ 0 = b' \text{ contr } \bot, \\ a' = b' \text{ contr } a = b. \end{cases}$$

It is easy to verify that the usual axioms for number theory as given by Kleene 1952, for example, can be derived in this system.

When one is interested in the reduction of derivations, the present formulation of first order arithmetic has a definite advantage over the standard one. Suppose namely that the numerical term a reduces to b. We then want a derivation of the form

$$\begin{array}{c} \\ \vdots \\ C[0] \end{array} \quad \begin{array}{c} C[x] \\ \vdots \\ C[x'] \end{array} \Bigg/ \begin{array}{c} C[a] \\ \vdots \end{array}$$

to reduce to the derivation in which a has been replaced by b,

$$\begin{array}{c} \\ \vdots \\ C[0] \end{array} \quad \begin{array}{c} C[x] \\ \vdots \\ C[x'] \end{array} \Bigg/ \begin{array}{c} C[b] \\ \hline C[a] \\ \vdots \end{array} \qquad C[a] \text{ conv } C[b]$$

However, if arithmetic is formulated without a rule of formula conversion, this reduction cannot be carried out without inserting a logically complex derivation of $C[a]$ from $C[b]$, and this is a transformation which destroys the structure of the derivation to such an extent that the transformed derivation fails to be definitionally equal to the original one.

3.2.2. The translation of first order arithmetic into the theory of types proceeds as follows.

3.2.2.1. Translation of the language.

3.2.2.1.1. A numerical variable x is translated into a variable x^* of type N.

3.2.2.1.2. 0^* is taken to be the term 0 of type N.

3.2.2.1.3. $(a')^*$ is $s(a^*)$.

3.2.2.1.4. $(a+b)^*$ is $R(b^*, a^*, (\lambda x^*)(\lambda y^*)s(y^*))$.

3.2.2.1.5. $(a \cdot b)^*$ is $R(b^*, 0, (\lambda x^*)(\lambda y^*)(y+a)^*)$.

3.2.2.1.6. An equation $a = b$ is translated into $E(a^*, b^*)$ where $E(a,b)$ is defined to be

$$R(a, (\lambda z)R(z, \top, (\lambda x)(\lambda y)\bot),$$
$$(\lambda u)(\lambda v)(\lambda z)R(z, \bot, (\lambda x)(\lambda y)v(x)))(b)$$

which is a term of type V and hence a type that satisfies the schema

$$\begin{cases} E(0,0) \text{ red } \top, \\ E(s(a),0) \text{ red } \bot, \\ E(0,s(b)) \text{ red } \bot, \\ E(s(a),s(b)) \text{ red } E(a,b). \end{cases}$$

Note that the axioms $\bot \in V$ and $\top \in V$ which form part of the reflection principle are needed in order to prove that $E(a,b)$ is a term of type V and hence a type.

3.2.2.1.7. The translation of composite formulae runs as in the case of first order predicate logic, the propositional constant $\top$ being translated into N_1.

3.2.2.2. Translation of the derivations. In addition to the rules of inference of first order predicate logic already stated, we have to consider the axiom $\top$, the induction schema and the rule of formula conversion.

3.2.2.2.1. The axiom $\top$ is translated into the term 1 of the type N_1 which is $\top^*$.

3.2.2.2.2. Let us now turn to the induction schema,

$$\dfrac{\begin{array}{cc} & C[x] \\ \vdots & \vdots \\ C[0] & C[x'] \end{array}}{C[a]}$$

By the hypothesis of the induction on the length of the given derivation, we have found terms d^* and $e^*[x^*, y^*]$ of types $C[0]^*$ and $C[x']^*$, respectively, y^* being the

variable of type $C[x]^*$ that corresponds to the assumption $C[x]$. The translation of the derivation of $C[a]$ is defined to be $R(a^*, d^*, (\lambda x^*)(\lambda y^*)e^*[x^*, y^*])$ which, by the N-elimination rule, is a term of type $C^*[a^*]$ or, what amounts to the same, $C[a]^*$.

Note that the translation is such that the induction contractions in first order arithmetic (see, for example,Prawitz 1971)

$$\begin{cases} \dfrac{\begin{array}{cc} & C[x] \\ \vdots & \vdots \\ C[0] & C[x'] \end{array}}{C[0]} \quad \text{contr} \quad \begin{array}{c} \vdots \\ C[0] \end{array} \\[2ex] \dfrac{\begin{array}{cc} & C[x] \\ \vdots & \vdots \\ C[0] & C[x'] \end{array}}{C[a']} \quad \text{contr} \quad \dfrac{\begin{array}{cc} & C[x] \\ \vdots & \vdots \\ C[0] & C[x'] \end{array}}{\begin{array}{c} C[a] \\ \vdots \\ C[a'] \end{array}} \end{cases}$$

are mapped into the contractions

$$\begin{cases} R(0, d^*, (\lambda x^*)(\lambda y^*)e^*[x^*, y^*]) \ \text{ contr } \ d^*, \\ R(s(a^*), d^*, (\lambda x^*)(\lambda y^*)e^*[x^*, y^*]) \ \text{ contr } \\ \qquad\qquad e^*[a^*, R(a^*, d^*, (\lambda x^*)(\lambda y^*)e^*[x^*, y^*])] \end{cases}$$

in the theory of types.

3.2.2.2.3. If the term a^* of type A^* is the translation of the derivation of the premise of an application of the rule of formula conversion

$$\dfrac{\begin{array}{c} \vdots \\ A \end{array} \quad A \text{ conv } B}{B}$$

we can take the translation of the derivation of B to be the same term a^*, because A conv B implies A^* conv B^* and hence we can conclude that a^* is a term of type B^* by the rule of type conversion.

3.3. Intuitionistic arithmetic of finite type with the axiom of choice.

3.3.1. The formalization of this theory that we shall consider extends the system of first order arithmetic specified above and differs in certain respects from the ones given by Spector 1962, Tait 1967 and Troelstra 1971.

3.3.1.1. Types. 0 is a type, and, if σ and τ are types, so is $\sigma \to \tau$.

3.3.1.2. Terms.

3.3.1.2.1. A variable of type τ is a term of type τ.

3.3.1.2.2. 0 is a term of type 0, and, if a is a term of type 0, so is a'.

3.3.1.2.3. If x is a variable of type σ and $b[x]$ is a term of type τ, then $(\lambda x)b[x]$ is a term of type $\sigma \to \tau$.

3.3.1.2.4. If c, d and $e[x, y]$ are terms of types 0, τ and τ, respectively, x being a numerical variable and y a variable of type τ, then $R(c, d, (\lambda x)(\lambda y)e[x, y])$ is a term of type τ.

3.3.1.2.5. If a and b are terms of types σ and $\sigma \to \tau$, respectively, then $b(a)$ is a term of type τ.

3.3.1.3. Formulae are built up from numerical equations by means of propositional connection and quantification of variables of arbitrary finite type.

3.3.1.4. Rules of contraction.

$$(\lambda x)b[x](a) \text{ contr } b[a],$$

$$\begin{cases} R(0, d, (\lambda x)(\lambda y)e[x, y]) \text{ contr } d, \\ R(a', d, (\lambda x)(\lambda y)e[x, y]) \text{ contr } e[a, R(a, d, (\lambda x)(\lambda y)e[x, y])], \end{cases}$$

$$\begin{cases} 0 = 0 \text{ contr } \top, \\ a' = 0 \text{ contr } \bot, \\ 0 = b' \text{ contr } \bot, \\ a' = b' \text{ contr } a = b. \end{cases}$$

3.3.1.5. The rules of inference are those of intuitionistic first order arithmetic, except that the quantifier rules have to be extended in the obvious way to all finite types. In addition, there is the (intuitionistically valid) axiom of choice

$$\forall x \exists y C[x, y] \supset \exists f \forall x C[x, f(x)]$$

with x, y and f of arbitrary types σ, τ and $\sigma \to \tau$, respectively.

3.3.2. The translation of this theory into the theory of types proceeds as follows.

3.3.2.1. Translation of the types. We take 0^* to be N and $(\sigma \to \tau)^*$ to be $\sigma^* \to \tau^*$.

3.3.2.2. Translation of the terms.

3.3.2.2.1. A variable x of type τ is translated into a variable x^* of type τ^*.

3.3.2.2.2. 0^* is the term 0 of type N, and $(a')^*$ is $s(a^*)$.

3.3.2.2.3. $((\lambda x)b[x])^*$ is defined to be $(\lambda x^*)b^*[x^*]$.

3.3.2.2.4. $(R(c, d, (\lambda x)(\lambda y)e[x, y]))^*$ is defined to be $R(c^*, d^*, (\lambda x^*)(\lambda y^*)e^*[x^*, y^*])$.

3.3.2.2.5. $(b(a))^*$ is defined to be $b^*(a^*)$.

3.3.2.3. The translation of the formulae runs as in the case of first order arithmetic, the only novelty being that $(\forall x B[x])^*$ and $(\exists x B[x])^*$ with x of type τ are defined to be $(\Pi x^* \in \tau^*)B^*[x^*]$ and $(\Sigma x^* \in \tau^*)B^*[x^*]$, respectively.

3.3.2.4. The interpretation of the rules of inference is no more complicated than in the case of first order arithmetic, the quantifier rules of inference of higher type being treated just like those of ground type. There remains the axiom of choice, whose translation

$$(\Pi x^* \in \sigma^*)(\Sigma y^* \in \tau^*)C^*[x^*, y^*] \rightarrow$$
$$(\Sigma f^* \in \sigma^* \rightarrow \tau^*)(\Pi x^* \in \sigma^*)C^*[x^*, f^*(x^*)]$$

is but an instance of the axiom of choice in the theory of types which we proved in section 2.5.

3.3.3. When the law of the excluded middle $A \vee \sim A$ (or, equivalently, reductio ad absurdum $\sim\sim A \supset A$) is added to intuitionistic arithmetic of finite type *with the axiom of choice*, the resulting system contains full simple type theory. (See Spector 1962, for example, in the case when the function which exists by virtue of the axiom of choice and the species which exists by virtue of the comprehension axiom both have arguments of ground type.) Thus the proof theoretic strength of the system increases by a very large amount. Since intuitionistic arithmetic of finite type with the axiom of choice is a subsystem of the theory of types, the same holds for the latter theory. Consequently, the axiom schema $A + -A$ (or, equivalently, $--A \rightarrow A$) is not consistent *relative* to the theory of types, although, of course, it may be proved to be consistent by stronger means. In particular, there is no hope for the double negation interpretation (see Kolmogorov 1925, Gödel 1933 and Gentzen 1933) to work, the reason for this being that the axioms for $+$ and Σ are stronger than the usual intuitionistic axioms for $\vee$ and $\exists$.

3.4. Intuitionistic arithmetical analysis with the axiom of choice.

3.4.1. To the system of first order arithmetic we now add n-ary predicate variables $X, Y, \ldots$ for every $n = 0, 1, \ldots$. An atomic formula is either of the form $a = b$ or of the form $B(a_1, \ldots, a_n)$ where B is an n-ary predicate term and $a_1, \ldots, a_n$ are numerical terms. Formulae are built up from atomic ones by means of the propositional connectives and quantifiers of both first and second order. An n-ary predicate term is either an n-ary predicate variable or of the form $(\lambda x_1 \ldots x_n)B[x_1, \ldots, x_n]$ where $B[x_1, \ldots, x_n]$ is a formula which contains no bound predicate variables. The first order quantifier rules of inference are extended in the obvious way to the second order. Finally, we add the axiom of choice

$$\forall x \exists X C[x, X] \supset \exists Y \forall x C[x, Y(x)].$$

Here Y is a predicate variable of one argument more than X and $C[x, Y(x)]$ denotes the result of replacing every part of $C[x, X]$ of the form $X(a_1, \ldots, a_n)$, by $Y(x, a_1, \ldots, a_n)$.

3.4.2. The translation of this theory into the theory of types proceeds as in the case of first order arithmetic with the following additions.

3.4.2.1. An n-ary predicate variable X is translated into a variable X^* of type

$$\underbrace{N \to \ldots \to N}_{n} \to V$$

in the theory of types.

3.4.2.2. $B(a_1, \ldots, a_n)^*$ is defined to be $B^*(a_1^*, \ldots, a_n^*)$.

3.4.2.3. $\forall X B[X]$ and $\exists X B[X]$ are translated into

$(\Pi X^* \in N \to \ldots \to N \to V) B^*[X^*]$ and $(\Sigma X^* \in N \to \ldots \to N \to V) B^*[X^*]$, respectively.

3.4.2.4. $((\lambda x_1, \ldots, x_n) B[x_1, \ldots, x_n])^*$ is defined to be $(\lambda x_1^*) \ldots (\lambda x_n^*) B^*[x_1^*, \ldots, x_n^*]$ which is seen to be a term of type $N \to \ldots \to N \to V$ by repeated use of the reflection principle and the fact that the translations $E(a^*, b^*)$ and $B^*(a_1^*, \ldots, a_n^*)$ of the atomic formulae $a = b$ and $B(a_1, \ldots, a_n)$ are terms of type V.

3.4.2.5. The second order quantifier rules of inference are interpreted just like the first order ones.

3.4.2.6. The translation of the axiom of choice

$$(\Pi x^* \in N)(\Sigma X^* \in N \to \ldots \to N \to V) C^*[x^*, X^*] \to$$
$$(\Sigma Y^* \in N \to N \to \ldots \to N \to V)(\Pi x^* \in N) C^*[x^*, Y^*(x^*)]$$

is just an instance of the axiom of choice in the theory of types which we proved in section 2.5.

4. THE NORMALIZATION THEOREM AND ITS CONSEQUENCES.

4.1. Normalization theorem. *Every term reduces to a normal term.*

Since we have introduced constants of every closed type, it will be sufficient to prove normalization for closed terms. Suppose namely that $a[x_1, \ldots, x_n]$ is an open term which depends on the variables x_1, ..., x_n of types A_1, ..., $A_n[x_1, \ldots, x_{n-1}]$, respectively. For $m = 1$, ..., n we may then introduce a constant a_m of the closed type $A_m[a_1, \ldots, a_{m-1}]$. By substituting the constants a_1, ..., a_n for the variables x_1, ..., x_n, we get a closed term which behaves just like $a[x_1, \ldots, x_n]$ from the point of view of normalization.

My proof of normalization uses an extension of the method of computability introduced by Tait 1967 in order to prove normalization for the terms of Gödel's 1958 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 types and the terms are generated separately from each other. This makes it possible, first, to define by induction on the construction of a type the notion of computability for terms of that type, and, second, to prove by induction on the construction of a term that it is computable. In the present theory, however, the definition of the notion of computability and the proof that an arbitrary term is computable can no longer be separated, because

the terms and the types are generated simultaneously. Instead, we have to show by induction on the construction of a type or term, in case A is a type, how to define the predicate φ_A which expresses the computability of a term of type A, and, in case a is a term of type A, that φ_A is defined and that $\varphi_A(a)$ holds, that is, that a is a computable term of type A.

The situation is further complicated by the fact that a type $A[x_1, \ldots, x_n]$ as well as a term $a[x_1, \ldots, x_n]$ of type $A[x_1, \ldots, x_n]$ in general depends on certain free variables x_1, ..., x_n of types A_1, ..., $A_n[x_1, \ldots, x_{n-1}]$, respectively. By induction hypothesis, we shall then know that φ_{A_1} has been defined and that, if a_1 is a closed term of type A_1 such that $\varphi_{A_1}(a_1)$, then $\varphi_{A_2[a_1]}$ has been defined and that, ..., if a_1, ..., a_{n-1} are closed terms of types A_1, ..., $A_{n-1}[a_1, \ldots, a_{n-2}]$, respectively, such that $\varphi_{A_1}(a_1)$, ..., $\varphi_{A_{n-1}[a_1,\ldots,a_{n-2}]}(a_{n-1})$ then $\varphi_{A_n[a_1,\ldots,a_{n-1}]}$ has been defined. Letting a_1, ..., a_n be closed terms of types $A_1, \ldots, A_n[a_1, \ldots, a_{n-1}]$, respectively, such that

$$\varphi_{A_1}(a_1),\ \ldots,\ \varphi_{A_n[a_1,\ldots,a_{n-1}]}(a_n),$$

we have to show, in case $A[x_1, \ldots, x_n]$ is a type, how to define

$$\varphi_{A[a_1,\ldots,a_n]}$$

and, in case $a[x_1, \ldots, x_n]$ is a term of type $A[x_1, \ldots, x_n]$, that $\varphi_{A[a_1,\ldots,a_n]}$ is defined and that

$$\varphi_{A[a_1,\ldots,a_n]}(a[a_1, \ldots, a_n])$$

holds, that is, that $a[a_1, \ldots, a_n]$ is a computable term of type $A[a_1, \ldots, a_n]$. Several cases have to be distinguished, one for each of the rules of type and term formation. In order to alleviate the notational burden, I shall not exhibit explicitly any free variables except the eigenvariables of the particular rule of type or term formation which is being considered.

It will be convenient to say that a term has *introduction* or *elimination form* depending on whether it has been formed by means of one of the introduction or one of the elimination rules. Thus, unless it is a constant, a closed term necessarily has either introduction or elimination form.

4.1.1. Definition of φ_A for a type symbol A.

4.1.1.1. $\varphi_{P(a_1,\ldots,a_n)}$ is the species of normalizable closed terms of type $P(a_1, \ldots, a_n)$.

4.1.1.2. Suppose that φ_A has been defined and that $\varphi_{B[a]}$ has been defined for all closed terms a of type A such that $\varphi_A(a)$. We then define $\varphi_{(\Pi x \in A)B[x]}$ by the following three clauses.

4.1.1.2.1. If $(\lambda x)b[x]$ is a closed term of type $(\Pi x \in A)B[x]$ and $\varphi_{B[a]}(b[a])$ for all closed terms a of type A such that $\varphi_A(a)$, then $\varphi_{(\Pi x \in A)B[x]}((\lambda x)b[x])$.

4.1.1.2.2. A closed normal term of type $(\Pi x \in A)B[x]$ which is not of the form $(\lambda x)b[x]$ satisfies $\varphi_{(\Pi x \in A)B[x]}$.

4.1.1.2.3. If the closed term b of type $(\Pi x \in A)B[x]$ has elimination form and reduces to a term a such that $\varphi_{(\Pi x \in A)B[x]}(a)$, then $\varphi_{(\Pi x \in A)B[x]}(b)$.

4.1.1.3. Again, suppose that φ_A has been defined and that $\varphi_{B[a]}$ has been defined for all closed terms a of type A such that $\varphi_A(a)$. We then define $\varphi_{(\Sigma x\in A)B[x]}$ by the following three clauses.

4.1.1.3.1. If a and b are closed terms of types A and $B[a]$, respectively such that $\varphi_A(a)$ and $\varphi_{B[a]}(b)$, then $\varphi_{(\Sigma x\in A)B[x]}((a,b))$.

4.1.1.3.2. A closed normal term of type $(\Sigma x\in A)B[x]$ which is not of the form (a,b) satisfies $\varphi_{(\Sigma x\in A)B[x]}$.

4.1.1.3.3. If the closed term b of type $(\Sigma x\in A)B[x]$ has elimination form and reduces to a term a such that $\varphi_{(\Sigma x\in A)B[x]}(a)$, then $\varphi_{(\Sigma x\in A)B[x]}(b)$.

4.1.1.4. Supposing φ_A and φ_B have been defined already, we define φ_{A+B} by the following three clauses.

4.1.1.4.1. If a is a closed term of type A such that $\varphi_A(a)$, then $\varphi_{A+B}(i(a))$. Similarly, if b is a closed term of type B such that $\varphi_B(b)$, then $\varphi_{A+B}(j(b))$.

4.1.1.4.2. A closed normal term of type $A+B$ which is neither of the form $i(a)$ nor of the form $j(b)$ satisfies φ_{A+B}.

4.1.1.4.3. If the closed term b of type $A+B$ has elimination form and reduces to a term a such that $\varphi_{A+B}(a)$, then $\varphi_{A+B}(b)$.

4.1.1.5. The predicate φ_V is defined by transfinite induction. But, simultaneously with the definition of the meaning of $\varphi_V(A)$, that is, of what constitutes a proof of $\varphi_V(A)$, we have to define by transfinite induction the predicate φ_A which expresses what it means for a closed term of the small type A to be computable. Actually, φ_A depends not only on A but also on the proof of $\varphi_V(A)$ although my notation does not indicate that explicitly.

4.1.1.5.1. If C is a closed normal term of type V which is not of the form $(\Pi x\in A)B[x]$, $(\Sigma x\in A)B[x]$ or $A+B$, then $\varphi_V(C)$ and φ_C is defined to be the species of normalizable closed terms of type C.

4.1.1.5.2. If $\varphi_V(A)$ and $\varphi_V(B[a])$ for all closed terms a of type A such that $\varphi_A(a)$, then $\varphi_V((\Pi x\in A)B[x])$ and $\varphi_{(\Pi x\in A)B[x]}$ is defined as in 4.1.1.2.

4.1.1.5.3. This case is like the previous one, replacing Π by Σ and referring to 4.1.1.3 instead.

4.1.1.5.4. If $\varphi_V(A)$ and $\varphi_V(B)$, then $\varphi_V(A+B)$ and φ_{A+B} is defined as in 4.1.1.4.

4.1.1.5.5. If the closed term B of type V has elimination form and reduces to a term A such that $\varphi_V(A)$, then $\varphi_V(B)$ and φ_B is set equal to φ_A.

4.1.1.6. If A is a term of type V such that $\varphi_V(A)$, then φ_A is the associated predicate defined in 4.1.1.5.

4.1.2. Lemma. *When defined, the predicate φ_A has the following three properties. First, $\varphi_A(a)$ holds if a is a closed normal term of type A which does not have introduction form. Second, if b is a closed term of type A which has elimination form and reduces to a term a such that $\varphi_A(a)$, then $\varphi_A(b)$. Third, $\varphi_A(a)$ implies that a is normalizable.*

We prove the lemma by induction on the definition of φ_A.

4.1.2.1. $\varphi_{P(a_1,\ldots,a_n)}$ is the species of normalizable closed terms of type $P(a_1,\ldots,a_n)$ and has therefore trivially the three properties stated in the lemma.

4.1.2.2. $\varphi_{(\Pi x\in A)B[x]}$ has trivially the first two properties. To verify the third, suppose that a term satisfies $\varphi_{(\Pi x\in A)B[x]}$. Then it reduces to a term which is either normal, in which case we are done, or else has the form $(\lambda x)b[x]$ and the property that $\varphi_{B[a]}(b[a])$ for all a such that $\varphi_A(a)$. By induction hypothesis, $\varphi_A(a)$ holds if a is a constant of type A. Consequently, $\varphi_{B[a]}$ is defined and $\varphi_{B[a]}(b[a])$ so that $b[a]$ is normalizable by induction hypothesis. And, $b[a]$ being normalizable, so is $(\lambda x)b[x]$.

4.1.2.3. $\varphi_{(\Sigma x\in A)B[x]}$ was defined so as to have the first two properties. To verify the third, suppose that a term satisfies $\varphi_{(\Sigma x\in A)B[x]}$. It must then reduce to a term which is either normal, in which case we are done, or else has the form (a,b) where $\varphi_A(a)$ and $\varphi_{B[a]}(b)$. By induction hypothesis, a and b are normalizable and, consequently, so is (a,b).

4.1.2.4. φ_{A+B} was defined so as to have the first two properties. To verify the third, suppose that a term satisfies φ_{A+B}. It must then reduce to a term which is either normal, in which case we are done, or else has the form $i(a)$ or $j(b)$ where, in the first case, $\varphi_A(a)$ and, in the second case, $\varphi_B(b)$. By induction hypothesis, $\varphi_A(a)$ implies that a is normalizable and $\varphi_B(b)$ implies that b is normalizable. Hence $i(a)$ is normalizable in the first case and $j(b)$ in the second.

4.1.2.5. φ_V was defined so as to have the first two properties. By transfinite induction on the proof of $\varphi_V(A)$, we shall at the same time prove that $\varphi_V(A)$ implies that A is normalizable and verify that the associated predicate φ_A has all the three properties stated in the lemma.

4.1.2.5.1. If C is a closed normal term of type V which is not of the form $(\Pi x\in A)B[x]$, $(\Sigma x\in A)B[x]$ or $A+B$, then C is a fortiori normalizable and the associated predicate φ_C, being defined as the species of normalizable closed terms of type C, has trivially all the three properties stated in the lemma.

4.1.2.5.2. Suppose that $\varphi_V((\Pi x\in A)B[x])$ is concluded from $\varphi_V(A)$ and $\varphi_V(B[a])$ for all terms a such that $\varphi_A(a)$. By induction hypothesis, A is normalizable and $\varphi_A(a)$ if a is a constant of type A. Hence $\varphi_V(B[a])$ so that, again by induction hypothesis, $B[a]$ is normalizable. But then so is $(\Pi x\in A)B[x]$. The verification that the lemma holds for $\varphi_{(\Pi x\in A)B[x]}$ is as in case 4.1.2.2.

4.1.2.5.3. This case is like the previous one, replacing Π by Σ and referring to 4.1.2.3 instead.

4.1.2.5.4. Suppose that $\varphi_V(A+B)$ has been concluded from $\varphi_V(A)$ and $\varphi_V(B)$. By induction hypothesis, A and B are normalizable and hence so is $A+B$. The verification that the lemma holds for φ_{A+B} is as in case 4.1.2.4.

4.1.2.5.5. Suppose that $\varphi_V(B)$ is concluded from $\varphi_V(A)$ and the knowledge that the closed term B of type V has elimination form and reduces to A. By induction hypothesis, A is normalizable and φ_A has all the three properties stated in the lemma. Hence B is normalizable and, since φ_B was set equal to φ_A, the lemma holds for φ_B as well.

4.1.2.6. That the lemma holds for the predicate φ_A associated with a term of type V such that $\varphi_V(A)$ has just been proved in 4.1.2.5. The proof of the lemma is now complete.

4.1.3. Lemma. *If φ_A and φ_B are both defined and A conv B, then $\varphi_A(a)$ if and only if $\varphi_B(a)$.*

Note that, in accordance with the remark in 4.1.1.5, the lemma is not trivially true even if A and B are syntactically equal, because even for one and the same type symbol A there may be different ways of defining the predicate φ_A.

Since A conv B, the types A and B are either both small or both large. In the latter case, they must be built up in the same way from V, definitionally equal atomic types of the form $P(a_1, \ldots, a_n)$ and definitionally equal small types. $\varphi_{P(a_1,\ldots,a_n)}$ was defined in 4.1.1.1 to be the species of normalizable closed terms of type $P(a_1, \ldots, a_n)$. Hence, if $P(a_1, \ldots, a_n)$ conv $P(b_1, \ldots, b_n)$, then $\varphi_{P(a_1,\ldots,a_n)}$ and $\varphi_{P(b_1,\ldots,b_n)}$ are extensionally equal because of the rule of type conversion. It now only remains to prove the lemma for two small types A and B. When A is a small type, φ_A is defined if and only if $\varphi_V(A)$. Therefore we can use transfinite induction on the proofs of $\varphi_V(A)$ and $\varphi_V(B)$. Several cases have to be distinguished depending on how $\varphi_V(A)$ and $\varphi_V(B)$ have been inferred.

4.1.3.1. If both $\varphi_V(A)$ and $\varphi_V(B)$ hold by virtue of 4.1.1.5.1, then φ_A and φ_B are the species of normalizable closed terms of types A and B, respectively, and so they are extensionally equal by the rule of type conversion.

4.1.3.2. If A and B have the forms $(\Pi x \in C)D[x]$ and $(\Pi x \in E)F[x]$, respectively, then C conv E and $D[x]$ conv $F[x]$. Hence φ_C and φ_D are extensionally equal by induction hypothesis. For the same reason, $\varphi_{D[c]}$ and $\varphi_{F[c]}$ are extensionally equal for all c such that $\varphi_C(c)$ or, equivalently, $\varphi_E(c)$. Being defined by 4.1.1.2, φ_A and φ_B are extensionally equal as well.

4.1.3.3. This case is like the previous one, replacing Π by Σ and referring to 4.1.1.3 instead.

4.1.3.4. If A and B have the forms $C + D$ and $E + F$, respectively, then C conv E and D conv F. Hence, on the one hand, φ_C and φ_E and, on the other hand, φ_D and φ_F are extensionally equal by induction hypothesis. Being defined by 4.1.1.4, φ_A and φ_B are extensionally equal as well.

4.1.3.5. If one of $\varphi_V(A)$ and $\varphi_V(B)$, say $\varphi_V(A)$, is inferred by 4.1.1.5.5, then A reduces to C such that $\varphi_V(C)$. By induction hypothesis, φ_C and φ_B are extensionally equal, and, since φ_A in this case is set equal to φ_C, so are φ_A and φ_B.

4.1.4. Verification that, if a is a term of type A, then φ_A is defined and $\varphi_A(a)$.

4.1.4.1. When we introduce a variable x of type A, we know by induction hypothesis that φ_A is defined. We have to show that φ_A is defined and that, if a is a closed term of type A, such that $\varphi_A(a)$, then $\varphi_A(a)$. This requires no argument.

4.1.4.2. When we introduce a constant a of type A, we know by induction hypothesis that φ_A is defined. We have to show that φ_A is defined and that $\varphi_A(a)$ holds which follows from the first part of lemma 4.1.2.

4.1.4.3. Π-introduction. By induction hypothesis, we know that φ_A is defined and that, if a is a closed term of type A, then $\varphi_{B[a]}$ is defined and $\varphi_{B[a]}(b[a])$.

Hence $\varphi_{(\Pi x\in A)B[x]}$ is defined by 4.1.1.2 and $\varphi_{(\Pi x\in A)B[x]}((\lambda x)b[x])$ holds by virtue of 4.1.1.2.1 which is what we had to prove.

4.1.4.4. Π-elimination. By induction hypothesis, we know that φ_A and $\varphi_{(\Pi x\in A)B[x]}$ are defined and that $\varphi_A(a)$ and $\varphi_{(\Pi x\in A)B[x]}(b)$ both hold. Three cases have to be distinguished corresponding to the defining clauses of $\varphi_{(\Pi x\in A)B[x]}$.

4.1.4.4.1. b is of the form $(\lambda x)d[x]$ and $\varphi_{B[a]}(d[a])$ holds for all a such that $\varphi_A(a)$. Then $b(a)$ has elimination form and reduces to $d[a]$ so that $\varphi_{B[a]}(b(a))$ by the second part of lemma 4.1.2.

4.1.4.4.2. b is normal and not of the form $(\lambda x)d[x]$. Let c be the normal form of a which exists by the third part of lemma 4.1.2. Then $b(a)$ reduces to $b(c)$ which is normal and has elimination form so that $\varphi_{B[a]}(b(a))$ by the first and second parts of lemma 4.1.2.

4.1.4.4.3. b has elimination form and reduces to d which satisfies $\varphi_{(\Pi x\in A)B[x]}$. Then $b(a)$ reduces to $d(a)$ which we have already shown to satisfy $\varphi_{B[a]}$. Hence $\varphi_{B[a]}(b(a))$ by the second part of lemma 4.1.2.

4.1.4.5. Σ-introduction. By induction hypothesis, φ_A is defined and $\varphi_{B[a]}$ is defined for all a such that $\varphi_A(a)$. Also, $\varphi_A(a)$ and $\varphi_{B[a]}(b)$. Hence $\varphi_{(\Sigma x\in A)B[x]}$ is defined by 4.1.1.3 and $\varphi_{(\Sigma x\in A)B[x]}((a,b))$ holds by virtue of 4.1.1.3.1.

4.1.4.6. Σ-elimination. By induction hypothesis, we know that $\varphi_{(\Sigma x\in A)B[x]}$ is defined and that $\varphi_{C[c]}$ is defined for all c such that $\varphi_{(\Sigma x\in A)B[x]}(c)$. Also, $\varphi_{(\Sigma x\in A)B[x]}(c)$ holds and $\varphi_{C[(a,b)]}(d[a,b])$ holds for all a and b such that $\varphi_A(a)$ and $\varphi_{B[a]}(b)$. Three cases have to be distinguished corresponding to the defining clauses of $\varphi_{(\Sigma x\in A)B[x]}$.

4.1.4.6.1. c has the form (a,b) where $\varphi_A(a)$ and $\varphi_{B[a]}(b)$. Then $E(c,(\lambda x)(\lambda y)$ $d[x,y])$ has elimination form and reduces to $d[a,b]$ which satisfies $\varphi_{C[(a,b)]}$. Hence $\varphi_{C[c]}(E(c,(\lambda x)(\lambda y)d[x,y]))$ by the second part of lemma 4.1.2.

4.1.4.6.2. c is normal and not of the form (a,b). Let a and b be constants of types A and $B[a]$, respectively. Then $\varphi_A(a)$ and $\varphi_{B[a]}(b)$ by the first part of lemma 4.1.2. Hence $\varphi_{C[(a,b)]}(d[a,b])$ so that $d[a,b]$ reduces to a normal term $g[a,b]$ by the third part of lemma 4.1.2. But then $E(c,(\lambda x)(\lambda y)d[x,y])$ reduces to $E(c,(\lambda x)(\lambda y)g[x,y])$ which is normal and has elimination form so that $\varphi_{C[c]}(E(c,(\lambda x)(\lambda y)d[x,y]))$ by the first two parts of lemma 4.1.2.

4.1.4.6.3. c has elimination form and reduces to a term f which satisfies $\varphi_{(\Sigma x\in A)B[x]}$. Then $E(c,(\lambda x)(\lambda y)d[x,y])$ reduces to $E(f,(\lambda x)(\lambda y)d[x,y])$ which we have already shown to satisfy $\varphi_{C[f]}$. Hence $\varphi_{C[c]}(E(c,(\lambda x)(\lambda y)d[x,y]))$ by the second part of lemma 4.1.2 and lemma 4.1.3.

4.1.4.7. $+$-introduction. By induction hypothesis, φ_A and φ_B are both defined and $\varphi_A(a)$ holds. Hence φ_{A+B} is defined by 4.1.1.4 and $\varphi_{A+B}(i(a))$ holds by virtue of 4.1.1.4.1. The second rule of $+$-introduction is treated in the same way.

4.1.4.8. $+$-elimination. By induction hypothesis, we know that φ_{A+B} is defined and that $\varphi_{C[c]}$ is defined for all c such that $\varphi_{A+B}(c)$. Also, $\varphi_{A+B}(c)$ holds and $\varphi_{C[i(a)]}(d[a])$ and $\varphi_{C[j(b)]}(e[b])$ hold for all a and b such that $\varphi_A(a)$ and $\varphi_B(b)$, respectively. Three cases have to be distinguished corresponding to

the defining clauses of φ_{A+B}.

4.1.4.8.1. c has the form $i(a)$ and $\varphi_A(a)$. Then $D(c,(\lambda x)d[x],(\lambda y)e[y])$ has elimination form and reduces to $d[a]$ which satisfies $\varphi_{C[i(a)]}$. Hence $\varphi_{C[c]}(D(c,(\lambda x)d[x],(\lambda y)e[y]))$ by the second part of lemma 4.1.2. The case when c is of the form $j(b)$ is treated in the same way.

4.1.4.8.2. c is normal and not of the form $i(a)$ or $j(b)$. Let a and b be constants of types A and B, respectively. Then $\varphi_A(a)$ and $\varphi_B(b)$ by the first part of lemma 4.1.2. Hence $\varphi_{C[i(a)]}(d[a])$ and $\varphi_{C[j(b)]}(e[b])$ so that $d[a]$ and $e[b]$ reduce to normal terms $g[a]$ and $h[b]$ by the third part of lemma 4.1.2. But then $D(c,(\lambda x)d[x],(\lambda y)e[y])$ reduces to $D(c,(\lambda x)g[x],(\lambda y)h[y])$ which is normal and has elimination form so that $\varphi_{C[c]}(D(c,(\lambda x)d[x],(\lambda y)e[y]))$ by the first two parts of lemma 4.1.2.

4.1.4.8.3. c has elimination form and reduces to a term f which satisfies φ_{A+B}. Then $D(c,(\lambda x)d[x],(\lambda y)e[y])$ reduces to $D(f,(\lambda x)d[x],(\lambda y)e[y])$ which we have already shown to satisfy $\varphi_{C[f]}$. Hence $\varphi_{C[c]}(D(c,(\lambda x)d[x],(\lambda y)e[y]))$ by the second part of lemma 4.1.2 and lemma 4.1.3.

4.1.4.9. N_n-introduction. φ_{N_n} is defined by 4.1.1.5.1 to be the species of normalizable closed terms of type N_n. Hence $\varphi_{N_n}(1)$, ..., $\varphi_{N_n}(n)$.

4.1.4.10. N_n-elimination. By induction hypothesis, we know that $\varphi_{C[c]}$ is defined for all c such that $\varphi_{N_n}(c)$. Also, $\varphi_{N_n}(c)$ and $\varphi_{C[1]}(c_1)$, ..., $\varphi_{C[n]}(c_n)$. We distinguish three cases depending on the form of c.

4.1.4.10.1. c is m. Then $R_n(c,c_1,\ldots,c_n)$ has elimination form and reduces to c_m which satisfies $\varphi_{C[m]}$. Hence $\varphi_{C[c]}(R_n(c,c_1,\ldots,c_n))$ by the second part of lemma 4.1.2.

4.1.4.10.2. c is normal and not one of 1, ..., n. By the third part of lemma 4.1.2, c_1, ..., c_n reduce to normal terms f_1, ..., f_n. Hence $R_n(c,c_1,\ldots,c_n)$ reduces to $R_n(c,f_1,\ldots,f_n)$ which is normal and has elimination form so that $\varphi_{C[c]}(R_n(c,c_1,\ldots,c_n))$ by the first two parts of lemma 4.1.2.

4.1.4.10.3. c is not normal but reduces to a normal term f. Then $R_n(c,c_1,\ldots,c_n)$ reduces to $R_n(f,c_1,\ldots,c_n)$ which we have already shown to satisfy $\varphi_{C[f]}$. Hence $\varphi_{C[c]}(R_n(c,c_1,\ldots,c_n))$ by the first part of lemma 4.1.2 and lemma 4.1.3.

4.1.4.11. N-introduction. φ_N is defined by 4.1.1.5.1 to be the species of normalizable closed terms of type N. Hence $\varphi_N(0)$ holds and $\varphi_N(a)$ implies $\varphi_N(s(a))$.

4.1.4.12. N-elimination. By induction hypothesis, we know that $\varphi_{C[a]}$ is defined for all a such that $\varphi_N(a)$. Also, $\varphi_N(c)$ and $\varphi_{C[0]}(d)$ hold and $\varphi_{C[s(a)]}(e[a,b])$ holds for all a and b such that $\varphi_N(a)$ and $\varphi_{C[a]}(b)$. We distinguish four cases depending on how we have inferred $\varphi_N(c)$.

4.1.4.12.1. c is 0. Then $R(c,d,(\lambda x)(\lambda y)e[x,y])$ reduces to d which satisfies $\varphi_{C[0]}$ so that $\varphi_{C[c]}(R(c,d,(\lambda x)(\lambda y)e[x,y]))$ by the second part of lemma 4.1.2.

4.1.4.12.2. c is of the form $s(a)$. We then know already that $\varphi_N(a)$ and $\varphi_{C[a]}(R(a,d,(\lambda x)(\lambda y)e[x,y]))$ both hold so that we can conclude $\varphi_{C[s(a)]}(d[a,R(a,d,(\lambda x)(\lambda y)e[x,y])])$. But $R(c,d,(\lambda x)(\lambda y)e[x,y])$ reduces to $d[a,R(a,d,(\lambda x)(\lambda y)e[x,y])]$ so that it must satisfy $\varphi_{C[c]}$ by the second part of lemma 4.1.2.

4.1.4.12.3. c is normal and has elimination form. From $\varphi_{C[0]}(d)$ we can conclude that d reduces to a normal term g by the third part of lemma 4.1.2. Also, let a and b be constant of types N and $C[a]$, respectively. Then $\varphi_N(a)$ and $\varphi_{C[a]}(b)$ by the first part of lemma 4.1.2. Hence $\varphi_{C[s(a)]}(e[a,b])$ so that $e[a,b]$ reduces to a normal term $h[a,b]$, again by the third part of lemma 4.1.2. But then $R(c,d,(\lambda x)(\lambda y)e[x,y])$ reduces to $R(c,g,(\lambda x)(\lambda y)h[x,y])$ which is normal and has elimination form so that $\varphi_{C[c]}(R(c,d,(\lambda x)(\lambda y)e[x,y]))$ by the first two parts of lemma 4.1.2.

4.1.4.12.4. c has elimination form and reduces to a term f such that $\varphi_N(f)$. Then $R(c,d,(\lambda x)(\lambda y)e[x,y])$ reduces to $R(f,d,(\lambda x)(\lambda y)e[x,y])$ which we have already shown to satisfy $\varphi_{C[f]}$. Hence $\varphi_{C[c]}(R(c,d,(\lambda x)(\lambda y)e[x,y]))$ by the second part of lemma 4.1.2 and lemma 4.1.3.

4.1.4.13. V-introduction. $\varphi_V(N_0)$, $\varphi_V(N_1)$, ... and $\varphi_V(N)$ all hold by virtue of definition 4.1.1.5.1. Next, suppose that $\varphi_V(A)$ holds and that $\varphi_V(B[a])$ holds for all a such that $\varphi_A(a)$. Then we can conclude $\varphi_V((\Pi x \in A)B[x])$ and $\varphi_V((\Sigma x \in A)B[x])$ by 4.1.1.5.2 and 4.1.1.5.3. Finally, suppose that $\varphi_V(A)$ and $\varphi_V(B)$ both hold. Then we can conclude $\varphi_V(A+B)$ by 4.1.1.5.4.

4.1.4.14. Type conversion. By induction hypothesis, φ_A and φ_B are both defined and $\varphi_A(a)$ holds. Hence $\varphi_B(a)$ by lemma 4.1.3. The proof of the normalization theorem is now complete.

4.2. Corollary. *Every type reduces to a normal type.*

Every type is built up by means of the operations Π, Σ, and $+$ from V, small types and atomic types of the form $P(a_1,\ldots,a_n)$. A small type, being a term of type V, is normalizable according to the normalization theorem, and so is a type of the form $P(a_1,\ldots,a_n)$ since a_1, ..., a_n are all terms. Hence every type is normalizable.

4.3. Corollary. *It can be mechanically decided whether or not two terms or two types are definitionally equal.*

Let A and B be two types. In order to decide whether or not A conv B we simply normalize A and B, which is possible according to the previous corollary, and check whether or not their normal forms are syntactically equal except possibly for differences in the naming of their bound variables. Similarly, if a is a term of type A and b a term of type B, we first decide whether or not A conv B and then, in case the answer is positive, whether or not a conv b. According to the normalization theorem, the latter decision can also be reached by normalizing a and b and checking if their normal forms are the same.

4.4. The form of the normal terms. In order to determine the syntactical form of the normal terms, it will be convenient to introduce some terminology. The *major subterm* of a term which has elimination form, is defined by stipulating that the major subterm of $b(a)$ is b and that the major subterm of $E(c,(\lambda x)(\lambda y)d[x,y])$, $D(c,(\lambda x)d[x],(\lambda y)e[y])$, $R_n(c,c_1,\ldots,c_n)$ and $R(c,d,(\lambda x)(\lambda y)e[x,y])$ in all cases is c. The *main redex* of a redex is the redex itself, and the main redex of a term which has elimination form but is not a redex is the main redex of its major subterm. If a term has elimination form,

then either it contains a main redex or else by taking the major subterm of its major subterm of ... of its major subterm we reach a constant or a free variable, because when we form a term of elimination form no free variable in its major subterm can become bound. In particular, a normal term must either have introduction form or contain a constant or a free variable. Hence we have proved (by purely combinatorial reasoning) that, in the system without object constants,

a closed normal term of type	must have the form
$(\Pi x \in A)B[x]$	$(\lambda x)b[x]$
$(\Sigma x \in A)B[x]$	(a, b)
$A + B$	$i(a)$ or $j(b)$
N_n	1, 2, ... or n
N	$s(s(\ldots s(0)\ldots))$
V	$(\Pi x \in A)B[x]$, $(\Sigma x \in A)B[x]$, $A + B$, N_0, N_1, ... or N

Combining this with the normalization theorem, we can conclude that a closed term of one of the types shown in the left column reduces to a term of the form exhibited on the same line in the right column.

4.5. Corollary. *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 of types does not require any proof at all for us, intelligent beings, who can understand the meaning of the types and the terms and recognize 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 of type $N \to N$ which denotes it. (Of course, f must not contain any object constants.) 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 of type N so that, according to what was proved in 4.4, 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 structure 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 construct in the formal theory can be computed by a machine with an arbitrary 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 1967 and 1970 has asked for. What is doubtful at present is not whether computerization is possible in principle, 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.

4.6. Completeness of intuitionistic first order predicate logic. Consider a first order formula C containing no other logical constants than $\supset$ and $\forall$, and let C^* be its translation into the theory of types as defined in section 3.1.2.1. Remember that in order to define the translation we had to introduce, first, a type constant I^* denoting the type of individuals, second, for every predicate constant P, a type constant P^* with all arguments of type I^*, and, third, for every function constant f, an object constant f^* of type $I^* \to \ldots \to I^* \to I^*$. We suppose that no other object constants than these have been introduced into the theory of types.

4.6.1. Theorem. *Let C be a closed first order formula. Then there is a closed term of type C^* in the theory of types if and only if C is provable in intuitionistic first order predicate logic.*

This shows that the fragment of first order predicate logic determined by $\supset$ and $\forall$ is complete *relative* to the theory of types. Kreisel 1970 has suggested to call this property *faithfulness* rather than completeness since it is quite different from the property that classical first order predicate logic enjoys by virtue of Gödel's completeness theorem.

The sufficiency was established already in section 3.1.2.2 where we showed how to translate a derivation c of a formula C in intuitionistic first order predicate logic into a term c^* of type C^* in the theory of types. The translation is such that c^* is closed if and only if the derivation c contains no free individual variables and no undischarged assumptions.

The necessity is a consequence of the normalization theorem and lemma 4.6.3 below.

4.6.2. Lemma. *Let a^* be a normal term of type I^* whose free variables are either of type I^* or of type A^* where A is a first order formula. Then there is an individual term a whose translation is a^*.*

The proof is by induction on the size of the term a^* which, being of type I^*, cannot have introduction form. Hence we can take the major subterm of ... of its major subterm until we reach either a variable or a constant. In the first case, it must be a variable x^* of type I^* and we are done, and, in the second case, it must be a constant f^* of type $I^* \to \ldots \to I^* \to I^*$ so that a^* is of the form $f^*(a_1^*, \ldots, a_n^*)$. By induction hypothesis, a_1^*, ..., a_n^* are translations of individual terms a_1, ..., a_n in first order logic. Hence a^* is the translation of the individual term $f(a_1, \ldots, a_n)$.

4.6.3. Lemma. *Let C be a first order formula and suppose that c^* is a normal term of type C^* whose free variables are either of type I^* or of type A^**

where A is a first order formula. Then there is a derivation c of the formula C in intuitionistic first order logic whose translation is c^*.

The proof is by induction on the construction of c^*. Three cases have to be distinguished.

4.6.3.1. c^* and C^* have the forms $(\lambda x^*)b^*[x^*]$ and $(\Pi x^* \in I^*)B^*[x^*]$, respectively. By induction hypothesis, there is a derivation

$$\begin{array}{c} \vdots \\ B[x] \end{array}$$

in intuitionistic first order logic whose translation is $b^*[x^*]$. Consequently, we can take c to be the derivation

$$\dfrac{\begin{array}{c} \vdots \\ B[x] \end{array}}{\forall x B[x]}$$

4.6.3.2. c^* and C^* have the forms $(\lambda x^*)b^*[x^*]$ and $A^* \to B^*$, respectively. By induction hypothesis, there is a derivation

$$\begin{array}{c} A \\ \vdots \\ B \end{array}$$

in intuitionistic first order logic whose translation is $b^*[x^*]$. Consequently, we can take c to be the derivation

$$\dfrac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{A \supset B}$$

in which the assumption A corresponding to the variable x^* of type A^* has been cancelled.

4.6.3.3. c has elimination form. Being normal, this is not possible unless it has the form $y^*(a_1^*, \ldots, a_n^*)$ where y^* is a variable of a type B^* which is the translation of a first order formula B. But then the type of a_i^* must be either I^* or of the form A_i^* where A_i is a first order formula. In the first case, we can conclude from the previous lemma that a_i^* must be the translation of an individual term a_i, and, in the second case, we can conclude from the induction hypothesis that a_i^* must be the translation of a derivation a_i of the formula A_i in intuitionistic first order logic. Consequently, c^* is the translation of the derivation c which is obtained by letting the assumption B be followed by a sequence of elimination inferences, in the first case, a $\forall$-elimination with the individual term a_i in the conclusion, and, in the second case, an application of modus ponens with A_i as minor premise.

Bibliography

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

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

Brouwer, L. E. J. (1918), 'Begründung der Mengenlehre unabhängig vom logischen Satz vom ausgeschlossenen Dritten. Erster Teil: Allgemeine Mengenlehre', *Verh. Nederl. Akad. Wetensch. Afd. Natuurk. Sect. 1* **12**(5), 3 – 43.

Curry, H. B. and Feys, R. (1958), *Combinatory Logic*, Vol. I, North-Holland, Amsterdam.

Feferman, S. (1969), Set-theoretical foundations of category theory, *in* 'Reports of the Midwest Category Theory Seminar III', Vol. 106 of *Lecture Notes in Mathematics*, Springer-Verlag, Berlin, pp. 201 – 247.

Gentzen, G. (1933), On the relation between intuitionistic and classical arithmetic, *in* E. Szabo, ed., 'The Collected Papers of Gerhard Gentzen', North-Holland, Amsterdam, 1969, pp. 53 – 67.

Gentzen, G. (1934), 'Untersuchungen über das logische Schliessen', *Math. Z.* **39**, 176 – 210 and 405 – 431.

Girard, J. Y. (1972), 'Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur', Thèse, Université Paris VII.

Goodman, N. D. (1970), A theory of constructions equivalent to arithmetic, *in* J. Myhill, A. Kino and R. E. Vesley, eds, 'Intuitionism and Proof Theory', North-Holland, Amsterdam, pp. 101 – 120.

Gödel, K. (1933), Zur intuitionistischen Arithmetik und Zahlentheorie, *in* 'Ergebnisse eines mathematischen Kolloquiums, Heft 4', pp. 34 – 38.

Gödel, K. (1958), 'Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes', *Dialectica* **12**, 280 – 287.

Howard, W. A. (1969), 'The formulae-as-types notion of construction'. Privately circulated notes.

Howard, W. A. (1972), 'A system of abstract constructive ordinals', *Journal of Symbolic Logic* **37**(2), 355 – 374.

Kleene, S. C. (1952), *Introduction to Metamathematics*, North-Holland, Amsterdam.

Kolmogorov, A. N. (1925), 'On the principle of excluded middle', *Mat. Sb.* **32**, 646 – 667.

Kreisel, G. (1962), Foundations of intuitionistic logic, *in* E. Nagel, P. Suppes and A. Tarski, eds, 'Logic, Methodology and Philosophy of Science', Stanford University Press, Stanford, California, pp. 198 – 210.

Kreisel, G. (1965), Mathematical logic, *in* T. L. Saaty, ed., 'Lectures on Mod-

ern Mathematics', Vol. III, John Wiley & Sons, New York, pp. 95 – 195.

Kreisel, G. (1968), Functions, ordinals, species, *in* B. van Rootselaar and J. F. Staal, eds, 'Logic, Methodology and Philosophy of Science III', North-Holland, Amsterdam, pp. 145 – 159.

Kreisel, G. (1970), 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, pp. 121 – 150.

Martin-Löf, P. (1971), Hauptsatz for the intuitionistic theory of iterated inductive definitions, *in* J. E. Fenstad, ed., 'Proceedings of the Second Scandinavian Logic Symposium', North-Holland, Amsterdam, pp. 179 – 216.

Prawitz, D. (1965), *Natural Deduction*, Almqvist & Wiksell, Stockholm.

Prawitz, D. (1971), Ideas and results in proof theory, *in* J. E. Fenstad, ed., 'Proceedings of the Second Scandinavian Logic Symposium', North-Holland, Amsterdam, pp. 235 – 307.

Russell, B. (1903), *The Principles of Mathematics*, Vol. I, Cambridge University Press, Cambridge.

Scott, D. (1970), Constructive validity, *in* 'Symposium on Automatic Demonstration', Vol. 125 of *Lecture Notes in Mathematics*, Springer-Verlag, Berlin, pp. 237 – 275.

Spector, C. (1962), Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics, *in* 'Recursive Function Theory', Vol. V of *Proceedings of Symposia in Pure Mathematics*, American Mathematical Society, Providence, Rhode Island, pp. 1 – 27.

Tait, W. W. (1967), 'Intensional interpretation of functionals of finite type', *Journal of Symbolic Logic* **32**, 198 – 212.

Tait, W. W. (1968), Constructive reasoning, *in* B. van Rootselaar and J. F. Staal, eds, 'Logic, Methodology and Philosophy of Science III', North-Holland, Amsterdam, pp. 185 – 199.

Troelstra, A. S. (1971), Notions of realizability for intuitionistic arithmetic and intuitionistic arithmetic in all finite types, *in* J. E. Fenstad, ed., 'Proceedings of the Second Scandinavian Logic Symposium', North-Holland, Amsterdam, pp. 369 – 405.