Proof-theoretic semantics for classical mathematics

W. W. Tait

The picture of mathematics as being about constructing objects of various sorts and proving the constructed objects equal or unequal is an attractive one, going back at least to Euclid. On this picture, what counts as a mathematical object is specified once and for all by effective rules of construction.

In the last century, this picture arose in a richer form with Brouwer's intuitionism. In his hands (for example, in his proof of the Bar Theorem), proofs themselves became constructed mathematical objects, the objects of mathematical study, and with Heyting's development of intuitionistic logic, this conception of proof became quite explicit. Today it finds its most elegant expression in the Curry-Howard theory of types, in which a proposition may be regarded, at least in principle, as simply a type of object, namely the type of its proofs. When we speak of 'proof-theoretic semantics' for mathematics, it is of course this point of view that we have in mind.

On this view, objects are given or constructed as objects of a ginen type. *That* an object is of this or that type is thus not a matter for discovery or proof. One consequence of this view is that equality of types must be a decidable relation. For, if an object is constructed as an object of type A and A and B are equal, then the object is of type B, too, and this must be determinable.

One pleasant feature of the type theoretic point of view is that the laws of logic are no longer 'empty': The laws governing the type

$$\forall x\!:\!A.F(x) = \Pi_{x\!A}F(x)$$

simply express our general notion of a function, and the laws governing

$$\exists x\!:\!A.F(x) = \Sigma_{x\!A}F(x)$$

express our notion of an ordered pair.

Much of my discussion applies equally to constructive mathematics. But the type-theoretic point of view remains, for many people, restricted to the domain of constructive mathematics. The term "classical" is included in the title to indicate that, on the contrary, classical mathematics can also be understood in this way and does not need to be founded on an inchoate picture of truth-functional semantics in the big-model-in-the-sky, a picture that can in any case never be coherently realized.

Of course, no particular system of types is going to capture all of classical—or for that matter, constructive—mathematics. In the classical case, the open-endedness can be expressed by the possibility of introducing ever larger systems of transfinite numbers as types. But here I will discuss only the elementary theory of types, omitting even the introduction of the finite numbers.

One thing to be noticed, and this is independent of whether or not one admits classical reasoning, is that Frege's simple ontology of function and object must be abandoned. In particular, his notion of a concept, as a truth-valued function, won't do: it must be replaced by the notion of a propositional or type-valued function. This is obvious in the case of constructive mathematics; but it applies equally to the classical case. What we prove are propositions, not truth-values: propositions may be said to *have* truth-values; but that is in virtue of being provable or refutable. Moreover, our concepts, i.e. proposition- or type-valued functionals, range not over the 'universe of all objects', as for Frege, but over specific types.

1. Another element of Frege's picture that I want to at least avoid is his notion of an 'incomplete object': the notion of a function as what is denoted by an open term and the notion of a propositional function as what is expressed by an open sentence. This very ugly idea has raised its head again in recent times because of the apparent need for bound variables in formulas such as

$$Qx : A.F(x)$$

where Q is a quantifier, and in terms

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

expressing universal quantifier introduction. ($x : A$ expresses the restriction of x to objects of type A.)

However useful in practice, variables are dispensible in the compositional semantics of type theory. Providing that we can always bring the formula $F(v)$ into the form $F'v$, where F' contains only variables in F other than v and denotes a propositional function defined on A, and similarly we can always bring $t(v)$ into the form $t'v$, where t' contains only variables in t other than v and denotes a function defined on A, then we may eliminate bound variables as primitive notation and write

$$Qx\!:\!A.F(x) \;:=\; QF' \qquad\qquad \lambda x\!:\!A.t(x) \;:=\; t'$$

That we can eliminate bound variables in this way I proved in [Tait 1998b], building on the work of Schönfinkel in [1924]. Let me describe the ontology upon which the elimination is founded.

More generally, consider a sentence of the form

$$Q_1x_1\!:\!A_1\ Q_2x_2\!:\!A_2\cdots Q_nx_n\!:\!A_n.F(x_1,\ldots,x_n)$$

where the Q_k are quantifiers and $F(x_1,\ldots,x_n)$ contains no bound variables. x_k is a variable of type A_k. In general, A_k may contain the variables x_j for $j<k$. Iterating the above procedure, we obtain

$$Q_1x_1\!:\!A'_1\ Q_2x_2\!:\!A'_2x_1\cdots Q_nx_n\!:\!A'_nx_1\cdots x_{n-1}.F'x_1\cdots x_n = Q_1\cdots Q_nF'$$

where

$$A'_kv_1\cdots v_{k-1} = A_k \qquad\qquad F'v_1\cdots v_n = F$$

The sequence

$$A'_1,\ldots,A'_n,F$$

is a type-base in the sense of the following

DEFINITION. The notion of a *type-base* or simply a *base* and of an *argument* for a base is defined by induction.

- The null sequence is a base and its only argument is the null sequence of objects.
- For $n\geq 0$, the sequence

$$F_0,\ldots,F_n$$

is a base iff $F_0, \ldots, F_{n-1}$ is a base and F_n is a type-valued function defined on the arguments of $F_0, \ldots, F_{n-1}$. An argument for this base is a sequence $b_0, \ldots, b_n$ such that $\mathbf{b} = b_0, \ldots, b_{n-1}$ is an argument for $F_0, \ldots, F_{n-1}$ and b_n is of type $F_n\mathbf{b}$. In particular, the unit sequence consisting of a type A is a base. Its arguments are the unit sequences consisting of objects of type A.

Thus, every initial segment $F_0, \ldots, F_k$ $(k < n)$ of a base $\mathbf{B} = F_0, \ldots, F_n$ is a base. If $b_0, \ldots, b_n$ is an argument for $\mathbf{B}$, then $F_k b_0 \cdots b_{k-1}$ is a type. We will call the terms in a base *functionals*. When $\mathbf{B}, F$ is a base, we call $\mathbf{B}$ the *base* of the functional F. When $\mathbf{B}$ is a base and $\mathbf{b}$ is an argument for it, we write

$$\mathbf{b}:\mathbf{B}$$

As a special case, when B is a type

$$b:B$$

means that b is an object of type B.

2. We assume given an initial stock of functionals, closed under bases, and, for each included type, all the objects of that type. We now extend this stock by means of base-forming operations. In §3, we extend the class of objects by means of object-forming operations

INSTANTIATION We may, à Schönfinkel, regard a functional of $n+1$ variables as a function of the first variable, whose values are functions of the remaining variables:

$$Fb_0b_1 \cdots b_n = (Fb_0)b_1 \cdots b_n$$

Thus, when $\mathbf{B} = A, F_0, \ldots, F_n$ is a base and $b:A$, then

$$\mathbf{B}b = F_0b, \ldots, F_nb$$

defines a base whose arguments are those sequences $b_0, \ldots, b_n$ such that $b, b_0, \ldots, b_n$ is an argument for $A, F_0, \ldots, F_n$. $\mathbf{B}b$ is called an *instantiation* of $\mathbf{B}$.

We assume, too, that when $\mathbf{B}$ is in the initial stock of bases, then so is $\mathbf{Bb}$.

QUANTIFICATION When F has base A, $\forall F$ and $\exists F$ are types.

We may extend quantification QF, defined initially for a functional whose base is a type, to arbitrary functionals with non-null bases as follows: when F has base $\mathbf{B}, G$, then QF has base $\mathbf{B}$. If $\mathbf{b}$ is an argument for $\mathbf{B}$, then $(QF)\mathbf{b}$ is defined point-wise by

$$(QF)\mathbf{b} = Q(F\mathbf{b})$$

If the base of the functional F is of length n, then we define its *universal closure* to be

$$F^* := \forall \cdots \forall F$$

with n occurrences of $\forall$. So F^* is a type.

If $b_0, \ldots, b_n$ is an argument for the base $F_0, \ldots, F_n$, then the type $F_k b_0 \cdots b_{k-1}$ of b_k depends upon $b_0, \ldots, b_{k-1}$. But sometimes we may wish to consider propositional functions F of n arguments of independent types $D_1, \ldots, D_n$, respectively. For example, in first or higher order predicate logic, we have variables ranging over the type of individuals, the type of sets of individuals, the type of sets of these, and so on. To deal with this, we need the following operation:

DUMMY ARGUMENTS. If A is a type and $F_0, \ldots, F_n$ is a base, the base $A, F_0[A], \ldots, F_n[A]$ is defined by

$$F_k[A]a = F_k$$

We extend this operation point-wise: If $\mathbf{B}, G$ and $\mathbf{B}, H_0, \ldots, H_n$ are bases, then so is

$$\mathbf{B}, G, H_0[G], \ldots, H_n[G]$$

where, for each argument $\mathbf{b}$ for $\mathbf{B}$

$$H_k[G]\mathbf{b} := H_k\mathbf{b}[G\mathbf{b}]$$

Now, given the list $D_1, \ldots, D_n$ of types, we may form the base $D^1, \ldots, D^n$ where $D^1 = D_1$ and

$$D^{k+1} = D_{k+1}[D^1][D^2]\cdots[D^k]$$

Then, if $\mathbf{b} = b_1, \ldots, b_n$ is an argument for this base, then

$$D^k b_1 \cdots b_{k-1} = D_k$$

Hence $\mathbf{b}$ is an argument for the base iff $b_k : D_k$ for each $k = 1, \ldots, n$.[1]

In terms of the quantifiers and dummy arguments, we can define implication and conjunction: Let F and G have base $\mathbf{B}$. then

$$F \longrightarrow G \;:=\; \forall G[F] \qquad F \wedge G \;:=\; \exists G[F]$$

Then $F \longrightarrow G$ and $F \wedge G$ have base $\mathbf{B}$. Note that, if $\mathbf{b} : \mathbf{B}$, then

$$(F \longrightarrow G)\mathbf{b} = (F\mathbf{b} \longrightarrow G\mathbf{b}) \qquad (F \wedge G)\mathbf{b} = (F\mathbf{b} \wedge G\mathbf{b})$$

Coimplication is defined between functionals with the same base by

$$F \longleftrightarrow G \;:=\; (F \longrightarrow G) \wedge (G \longrightarrow F)$$

There is one more operation on bases that we need, besides quantification, instantiation and introducing dummy arguments:

TRANSPOSITION. Let

$$F, G, H_0, \ldots, H_n$$

be a base. $\forall G$ is a type and $F[\forall G]$ has base $\forall G$; so $\forall G, F[\forall G]$ is a base. If cd is an argument for this base, then $c : \forall(G)$ and $d : F[\forall G]c = F$. So cd is

[1] We could avoid this admittedly artificial treatment of variables of independent types, but at the cost of a more complex structure of bases. Namely, we would define bases to be trees, where, besides the null base, trees of the form

$$\frac{\mathbf{B}_1 \quad \cdots \quad \mathbf{B}_n}{F}$$

are admitted as bases when $n \geq 0$, $\mathbf{B}_1, \ldots, \mathbf{B}_n$ are bases and, F is a type-valued function defined on $\mathbf{B}_1 \times \cdots \times \mathbf{B}_n$. An argument for this base is of the form $\mathbf{b}_1, \ldots, \mathbf{b}_n, c$, where $\mathbf{b}_k : \mathbf{B}_k$ for each $k = 1, \ldots, n$ and $c : F(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. But, in the interests of simplicity, if not in the interests of efficient computation, we will continue to deal only with linear bases.

defined and is of type Gd. Thus d, cd is an argument for F, G. It follows that we can form a new base

$$\forall G, F[\forall G], H_0\{G\}, \ldots, H_n\{G\}$$

where the functionals $H_k\{G\}$ are defined by

$$H_k\{G\}cd := H_k d(cd).$$

Again, we may extend the operation point-wise: Let $\mathbf{B}, F, G, H_0, \ldots, H_n$ be a base. Then the base $\mathbf{B}, \forall G, F[\forall G], H_0\{G\}, \ldots, H_n\{G\}$ is defined by

$$H_k\{G\}\mathbf{b} = H_k\mathbf{b}\{G\mathbf{b}\}$$

3. We turn now to object-forming operations.
The LAW OF $\forall$-ELIMINATION is

$$f\!:\!\forall F \;\;\&\;\; b\!:\!A \;\;\Rightarrow\;\; fb\!:\!Fb.$$

The LAW OF $\exists$-INTRODUCTION is

$$b\!:\!A \;\;\&\;\; c\!:\!Fb \;\;\Rightarrow\;\; (b,c)\!:\!\exists F$$

and the LAW OF $\exists$-ELIMINATION takes the form of projections:

$$p\!:\!\exists F \;\;\Rightarrow\;\; p1\!:\!A \;\;\&\;\; p2\!:\!F(p1).$$

Note that 1 and 2 do not count here as objects.[2]

In order to obtain the law of $\forall$-Introduction without using lambda-abstraction, we need to introduce a generalization of Schönfinkel's combinators.

COMBINATOR K. If A and B are types, then

$$K(A,B)\!:\!A \longrightarrow (B \longrightarrow A)$$

where

$$K(A,B)ab = a.$$

[2]This form of $\exists$-Elimination is different from that of Martin-Löf, e.g. in [Martin-Löf 1998], although, as he notes, the two forms are equivalent. It would seem that projection more directly expresses what it means to have an object of type $\exists F$.

This is the typed version of one of Schönfinkel's combinators.

We extend K to functionals F and G with a common base $\mathbf{B}$: if $\mathbf{b}:\mathbf{B}$, then

$$K(F,G):(F \longrightarrow (G \longrightarrow F))^*$$

is defined point-wise by setting

$$K(F,G)\mathbf{b} = K(F\mathbf{b}, G\mathbf{b})$$

COMBINATORS $S_\forall$ and $S_\exists$. Let H have base F, G and let Q be a quantifier $\forall$ or $\exists$. Then $\forall QH$ and $Q\forall(H\{G\})$ both are types. We will define the combinator

$$S_Q(H):(\forall QH \longrightarrow Q\forall(H\{G\})$$

Let $c:\forall Q(H)$. Then

$$S_Q(H)c:Q\forall(H\{G\})$$

$H\{G\}$ has base $\forall G, F[\forall G]$.

First, let $Q = \forall$.

$$S_\forall(H)c:\forall\forall(H\{G\})$$

Let $d:\forall G$ and $e:F = F[\forall G]c$. $S_\forall(H)cde$ must be defined to be of type $H\{G\}de$, i.e. of type $He(de)$, which is the type of $ce(de)$. Hence, we define

$$S_\forall(H)cde = ce(de)$$

Thus, $S_\forall$ is the typed version of Schönfinkel's other combinator.

Now let $Q = \exists$.

$$S_\exists(H)c:\exists\forall(H\{G\})$$

Thus we must have

$$S_\exists(H)c1:\forall G$$

$$S_\exists(H)c2 : \forall H\{G\}(S_\exists(H)c1)$$

Let $d : F$. Then we must have

$$S_\exists(H)c1d : Gd$$

$$S_\exists(H)c2d : Hd(S_\exists(H)c1d)$$

But $cd : \exists Hd$ and so $cd1 : Gd$ and $cd2 : Hd(cd1)$. So we may define $S_\exists(H)$ by

$$S_\exists(H)c1d := cd1 \qquad S_\exists(H)c2d := cd2$$

We again extend the combinators $S_Q(H)$ to the case in which H has a base $\mathbf{B}, F, G$ by point-wise definition:

$$S_Q(H) : (\forall QH \longrightarrow Q\forall H\{G\})^*$$

Let $\mathbf{b} : \mathbf{B}$. Then

$$S_Q(H)\mathbf{b} := S_Q(H\mathbf{b})$$

Notice that, if H has base $A, B[A]$, the type

$$\forall\exists H \longrightarrow \exists\forall H\{B[A]\}$$

of $S_\exists(H)$, which may be expressed by

$$\forall x : A\exists y : B.Hxy \longrightarrow \exists z : (A \longrightarrow B)\forall x : A.Hx(zx)$$

is an expression of the Axiom of Choice. Our definition of the combinator $S_\exists(H)$ amounts to a constructive proof of this axiom.

4. We need now to discuss the notion of definitional equality. We are discussing a system Σ of bases and objects built up by means of certain operations: instantiation, quantification, introducing dummy arguments, transposition, and $\exists$ and $\forall$ introduction and elimination from a given stock of bases and objects, which are distinct from the newly introduced ones. We assume that equality between given functionals or objects is given and we assume that it is closed under instantiation and $\forall$-elimination. Thus, besides the

defining equations given above for the new functionals and objects, we have all of the true equations

$$Fa = G \qquad fa = b$$

concerning given functionals F and G and given objects a and b (when f is of some type $\forall H$ and A, F is a given base). We can extend the equality relation to the new objects and types by taking it to be the equivalence relation generated by the defining equations and the true equations concerning the given objects and functionals. However, for functionals in general, we need a weaker (i.e. more inclusive) notion of equality, which we can define by induction on the length of their bases: two functionals F and G are equal iff their bases have the same length n and the members are pairwise equal and if $Fx_1 \cdots x_n = Gx_1 \cdots x_n$ follows purely formally from the given equations, where $x_1, \ldots, x_n$ are distinct symbols. We need to specify that, when an object c is of some type A and $A = B$, then c is of type B, too.

It can be shown that this definition of equality, called *definitional equality*, is decidable relative to the true equations concerning the given functionals and objects. It turns out, too, that when $b = c$ and $b{:}A$, then $c{:}A$. So each object has a unique type.

The inelegant definition of equality between functionals with non-null bases is necessitated by the fact that we need below some special cases of the equations

$$H[G]\{G\} = H[\forall G]$$

$$(QH)[G] = Q(H[G])$$

whenever Q is a quantifier and the left-hand sides are meaningful. These equations are valid for the notion of equality we have just introduced; but it would be more satisfactory to be able to extend this list of equations to a 'complete' one, i.e. so that equality of objects or functionals in general could be taken to be the equivalence relation generated by all the equations. For example, besides the above equations, we would need

$$(QH)\{G\} = Q(HG)$$

$$H[F][G[F]] = H[G][F]$$

$$H\{F\}\{G\{F\}\} = H\{G\}\{F\}$$

$$H[F]\{G[F]\} = H\{G\}[F]$$

when, again, the left-hand sides make sense.

QUESTION. Is this system of equations complete? I suspect so; but if not, how is it to be extended to a complete system?

5. Let G and H have base $\mathbf{B}, F$, so that $H[G]\{G\} = H[\forall G]$. Let $S = S_\forall(H[G])$. Then

$$S : (\forall\forall H[G] \longrightarrow \forall\forall(H[G]\{G\}))^*$$

Hence

$$S : (\forall(G \longrightarrow H) \longrightarrow \forall\forall(H[\forall G]))^*$$

But $\forall(H[\forall G]) = (\forall H)[\forall G]$ and so, finally

$$S : (\forall(G \longrightarrow H) \longrightarrow (\forall G \longrightarrow \forall H)) \tag{1}$$

Note that the types of $K(F, G)$ and S, i.e.

$$(F \longrightarrow \forall F[G])^* \qquad \text{and} \qquad (\forall(G \longrightarrow H) \longrightarrow (\forall G \longrightarrow \forall H))^*$$

are precisely Quine's [1951] axioms for the universal quantifier in first-order predicate logic, which have the property of avoiding λ-abstraction, i.e. hypothetical proof. The difference is that Quine retains bound variables in formulas. Both Quine [1960a] and Bernays [1959] discussed the question of eliminating bound variables in formulas in first-order logic; but neither presented an entirely adequate account from the point of view of proof theory, even for predicate logic.

If in the equation (1) we replace G and H by $B[A]$ and $C[A]$, respectively, where A, B, C are types, then we obtain Schönfinkel's combinator S of type

$$[A \longrightarrow (B \longrightarrow C)] \longrightarrow [(A \longrightarrow B) \longrightarrow (A \longrightarrow C)]$$

This type, together with the type of K(A,B):

$$A \longrightarrow (B \longrightarrow A)$$

are the axioms of positive implicational logic. This correspondance between the positive implicational logic and the typed theory of combinators seems to have been first noticed in [Curry Feys 1958] and is cited in [Howard 1980] as one of the sources of the propositions-as-types point of view.

6. So far, we've said nothing about eliminating bound variables, or what amounts to the same thing, eliminating the need for free variables. In order to show how we can eliminate variables, we first have to introduce them.

Let A be a type in Σ and let v be a symbol that we take as an indeterminate of type A. We can construct the formal 'polynomial extension' $\Sigma[v]$ of Σ in the obvious way, formally closing it under the above operations and where equality is again the relation of definitional equality. For every $b : A$ in Σ, there is a homomorphism $b^* : \Sigma[v] \longrightarrow \Sigma$ obtained by 'substituting b for v in the formulas and terms of $\Sigma[v]$'. Restricted to Σ, b^* is the identity function. The following is proved in [Tait 1998b]

EXPLICIT DEFINITION THEOREM. Let $F_1, \ldots, F_n$ be a base and t a term of type B in $\Sigma[v]$.

- There is a base $A, F'_1, \ldots, F'_n$ in Σ such that, for any $b : A$ in Σ,

$$F'_k b = F_k \qquad (k = 1, \ldots, n)$$

- There is an object $t' : \forall B'$ in Σ such that for all $:_{-} A$ in Σ

$$t'b = t$$

We can write

$$\lambda x : A.F(x) := F' \qquad\qquad \lambda x : A.t(x) := t'$$

This process of taking formal extensions can be iterated: let $\Sigma[v_0, \ldots, v_n]$ be given and let B_{n+1} be a type in this system. Choose a new symbol v_{n+1} as an indeterminate of type B and form $\Sigma[v_0, \ldots, v_{n+1}] = \Sigma[v_0, \ldots, v_n][v_{n+1}]$.

Given a functional $F(v_1, \ldots, v_n)$ in the system $\Sigma[v_1, \ldots, v_n]$, we can construct the functional

$$F' = \lambda x_1 : B_1 \cdots \lambda x_n : B_b.F(x_1, \ldots, x_n)$$

in Σ such that

$$F'v_1 \cdots v_n = F(v_1, \ldots, v_n).$$

7. Notice that we have not introduced disjunction in type theory: it is indeed an awkward operation. Were we to introduce the type $\mathbf{N}$ of the natural numbers with its corresponding introduction and elimination rules, the functional $= 0$ with base $\mathbf{N}$ can be defined and so disjunction can be defined by

$$A \vee B := \exists x{:}\mathbf{N}[(x = 0 \longrightarrow A) \wedge (x \neq 0 \longrightarrow B)]$$

But the most elementary way to deal with disjunction is to introduce the base

$$\mathbf{2}, \mathbf{T}$$

$\mathbf{2}$ is the two-object type. $\mathbf{2}$-Introduction specifies that the *truth-values* $\top$ and $\bot$ are of type $\mathbf{2}$. $\mathbf{2}$-Elimination asserts the existence, for any functional F with base $\mathbf{2}$, of

$$\mathbf{N}_2(F){:}[F\top \longrightarrow (F\bot \longrightarrow \forall F)]$$

where

$$\mathbf{N}_2(F)bc\top := b \qquad\qquad \mathbf{N}_2(F)bc\bot := c$$

We take $\mathbf{T}\top$ to be the terminal type and $\mathbf{T}\bot$ to be the initial type, i.e.

$$\mathbf{1} := \mathbf{T}\top \qquad\qquad \mathbf{0} := \mathbf{T}\bot$$

$\mathbf{1}$-Introduction specifies that ι is an object of type $\mathbf{1}$ and $\mathbf{1}$-Elimination asserts the existence, for any functional F of base $\mathbf{1}$, of

$$\mathbf{N}_1(F){:}[F\iota \longrightarrow \forall F]$$

where

$$\mathbf{N}_1(F)b\iota := b$$

There is no **0**-Introduction, of course. **0**-Elimination asserts the existence, for any functional F of base **0**, of

$$\mathbf{N}_0(F) : \forall F$$

Our types **k** are of course Martin-Löf's [1998] types N_k.

In order to preserve the Explicit Definition Theorem, the elimination rules for $\mathbf{k} = \mathbf{2}, \mathbf{1}$ and **0** need to be extended in the usual way by point-wise definition to functionals F with bases of the form

$$\mathbf{B}, \mathbf{k}[\mathbf{B}]$$

where, if **B** is $G_0, \ldots, G_n$, then $\mathbf{k}[\mathbf{B}] := \mathbf{k}[G_0][G_1] \cdots [G_n]$. We may do this as follows: Let F^+ with base $\mathbf{k}, \mathbf{B}[\mathbf{k}]$ be defined by

$$F^+ := \lambda x : \mathbf{k} \lambda y_1 : B_1 \cdots \lambda y_n : B_b . F y_1 \cdots y_n x$$

Then we define

$$\mathbf{N}_2(F) : [F^+ \top \longrightarrow (F^+ \bot \longrightarrow \forall F)]^*$$

$$\mathbf{N}_1(F) : [F^+ \iota \longrightarrow \forall F]^*$$

$$\mathbf{N}_0(F) : \forall F$$

by

$$\mathbf{N}_\mathbf{k}(F)\mathbf{b} := \mathbf{N}_\mathbf{k}(F\mathbf{b})$$

for $\mathbf{b} : \mathbf{B}$.

If F has base **B**, then we define

$$\neg F := F \longrightarrow \mathbf{0}[\mathbf{B}].$$

Let F and G have the base **B**. Set

$$\langle F, G \rangle := \exists x : \mathbf{2} . [(\mathbf{T}x \longrightarrow A) \wedge (\neg \mathbf{T}x \longrightarrow B)].$$

It is easy to deduce (i.e. find objects of type)

$$(2) \qquad (F \longleftrightarrow \langle F, G \rangle \top)^* \qquad (G \longleftrightarrow \langle F, G \rangle \bot)^*.$$

So we may define

$$F \vee G := \exists x : \mathbf{2} \langle F, G \rangle x.$$

With this definition, using the deducibility of (2), the usual laws of $\vee$-Introduction and -Elimination are derivable.

8. Up to now, we have been discussing the theory of types in general, with no particular reference to the classical theory. The latter is of course obtained by adding the law of

$\neg\neg$-ELIMINATION. Let F have base $\mathbf{B}$. Then

$$D(F) : (\neg\neg F \longrightarrow F)^*,$$

Here again, when $\mathbf{B}$ is non-null, $D(F)$ is defined point-wise in terms of $D(A)$ for A a type:

$$D(F)\mathbf{b} := D(F\mathbf{b})$$

when $\mathbf{b} : \mathbf{B}$.

From the type-theoretic point of view, what characterizes classical mathematics is not truth-functional semantics, but the introduction of what, *somewhat* in the spirit of Hilbert, we may call the *ideal* objects $D(F)$.

It is interesting that the problem that writers have found with classical mathematics, that there exist undecidable propositions A, such as the continuum hypothesis, for which the law of excluded middle nevertheless is taken to be valid, takes a different form when one moves from the (anyway incoherent) point of view of truth-functional semantics to that of type theory. Even constructively we can produce a deduction p of

$$\neg\neg(A \vee \neg A)$$

So classically we have

$$q = D(A \vee \neg A)))p : A \vee \neg A$$

I.e.

$$q : \exists x : \mathbf{2}.\langle A, \neg A \rangle x$$

Therefore, $q1 : \mathbf{2}$ and $q2 : \langle A, \neg A\rangle(q1)$. But we are unable to conclude that $q1$ is $\top$ and we are unable to conclude that it is $\bot$; and so we are unable to conclude that $q2$ is a proof of A or that it is a proof of $\neg A$. We will see below that, unlike the relation of definitional equality, we can define in type theory the notion of (extensional) equality $\equiv$. Hence, by **2**-Elimination (which states that anything true of both $\top$ and $\bot$ is true of all objects of type **2**)

$$q1 \equiv \top \ \vee \ q1 \equiv \bot$$

will be derivable, since

$$\top \equiv \top \ \vee \ \top \equiv \bot$$

and

$$\bot \equiv \top \ \vee \ \bot \equiv \bot$$

will both be derivable. Notice that this situation concerns not just undecidable propositions in classical mathematics, but *any* proposition A for which excluded middle cannot be proved constructively. Also, in the same way in which there are ideal objects of type **2** which cannot be said to be $\top$ and cannot be said to be $\bot$, but nevertheless are either one or the other, so there are ideal numbers for example, i.e. objects of type **N** (were we to introduce this type), which therefore are in the sequence $0, 1, 2 \ldots$ but which we cannot locate in this sequence. This is an interesting observation about how disjunction works in classical mathematics; but it seems paradoxical only when one assumes that classical mathematics is based on truth-functional semantics.

9. In classical logic, the sets of elements of type A are not themselves types, but are precisely the **2**-valued functions on A, i.e. they are the objects of type

$$\mathbf{P}(A) := A \longrightarrow \mathbf{2}$$

We may define the functional ϵ_A with base $A, \mathbf{P}(A)[A]$ by

$$a\epsilon_A f := \epsilon_A a f := T(fa)$$

So when $fa = \top$, $a\epsilon f$ is the true proposition **1** and when $fa = \bot$, it is the false proposition **0**.

We have shown how, in classical logic, for any functional F with base A, to obtain an object

$$p \,:\, \forall x{:}A[F(x) \vee \neg F(x)]$$

Let $f = \lambda x : A(px1)$ and $g = \lambda x : A(px2)$. Then $f : A \longrightarrow \mathbf{2} = \mathbf{P}(A)$. Let $u{:}A$. Then gu is a deduction of $\langle F(u), \neg F(u)\rangle(fu) =$

$$[\mathbf{T}(fu) \longrightarrow F(u)] \wedge [\neg\mathbf{T}(fu) \longrightarrow \neg F(u)]$$

From the second conjunct, in classical logic, we obtain a deduction of $F(u) \longrightarrow \mathbf{T}(fu)$. Recalling that $u\epsilon_A f$ is $\mathbf{T}(fu)$, we thus have a deduction $r(u)$ of

$$u\epsilon_A f \longleftrightarrow F(u)$$

So $(f, \lambda x{:}A.r(x))$ is a deduction of the *COMPREHENSION PRINCIPLE*[3]

$$\exists z{:}\mathbf{P}(A)\forall x{:}A[x\epsilon z \longleftrightarrow F(x)]$$

10. In this final section, I will show that extensional equality can be defined in the Curry-Howard theory.[4] Recall that equality between types is to be understood intensionally, as *definitional equality*. This is important for the type-theoretic point of view: what the type of an object is should be determined from the object. If c is given as an object of type A and A is the same type as B, then we should be able to determine that c is of type B. Similarly, the identity of objects should be understood in terms of definitional equality: objects are given by terms and two terms denote the same object iff they are definitionally equal. Of course, definitional equality as a relation among the objects of some type A is not expressible as a type; that is, there is no functional E with base $A, A[A]$ such that the type Ebc expresses the definitional equality of objects b and c of type A.

[3]Thus, the Comprehension Principle follows from the Law of Excluded Middle. The converse is also true: Let B be any type. Using the Comprehension Principle, there is a $b{:}\mathbf{P}(\mathbf{2})$ such that $\mathbf{T}(b\bot) \longleftrightarrow B$. Hence, $\neg\neg\mathbf{T}(b\bot) \longleftrightarrow \neg\neg B$. But even constructively, $\neg\neg\mathbf{T}(b\bot) \longrightarrow \mathbf{T}(b\bot)$ and so $\neg\neg B \longrightarrow B$.

[4]I discussed this in [1996]; but the treatment, besides being unnecessarily complicated, contained an error which was discovered in conversations with Robert Harper and Christopher Stone at Carnegie-Mellon in Autumn 1999. Part of the complication in the earlier paper resulted from the fact that I thought that extensional equality could be treated only in the classical system. We shall see that this is false.

There is, however, the notion of *extensional equality* between objects of a given type, which we can express in type theory. Let me discuss this.

An immediate problem with defining extensional equality between objects of the same type is this: Let c and d be of type $\forall x : A.F$. Then clearly the extensional equality of c and d should imply that, for all extensionally equal a and b of type A, ca should be extensionally equal to db. But the types Fa and Fb of ca and db, respectively, are not in general the same type, i.e., are not definitionaly equal. So we must define extensional equality between object of certain different types.

In virtue of the Explicit Definition Theorem, every functional is of the form

$$Fb_1 \cdots b_n$$

where $n \geq 0$ and F is built up without using instantiation. If

$$Fc_1 \cdots c_n$$

is another functional, then we call these two functionals *congruent*. Obviously, congruence is an equivalence relation on the functionals. We need to define the functional $\equiv_{FG}$ of extensional equality for congruent functionals F and G. We will drop the subscript on $\equiv_{FG}$ when no confusion results. Let F have base $\mathbf{B}$ and G base $\mathbf{B}'$. $\equiv_{FG}$ will be defined as a functional with base $\mathbf{B}, \mathbf{B}'[\mathbf{B}], F[\mathbf{B}'[\mathbf{B}]], G[\mathbf{B}, F[\mathbf{B}'[\mathbf{B}]]]$, point-wise: for each $\mathbf{b} : \mathbf{B}$ and $\mathbf{b}' : \mathbf{B}'$

$$\equiv_{FG} \mathbf{b}\mathbf{b}' \quad := \quad \equiv_{F\mathbf{b}G\mathbf{b}'} .$$

So it suffices to define $\equiv_{AB}$ for congruent types A and B. But for this we need only define what $a \equiv b$ means for objects $a : A$ and $b : B$ in some polynomial extension of Σ. For then we obtain $\equiv$ as $\lambda x : A \lambda y : B.x \equiv y$.

We assume that the relation of extensional equality is defined for the basic types and that it is an equivalence relation. We define it now for the new types that we have introduced.

DEFINITION OF *EXTENSIONAL EQUALITY* Let $a : A$ and $b : B$.

- $A = B = \mathbf{2}$. Recalling that $\mathbf{T}\top$ is the terminal type $\mathbf{1}$ and $\mathbf{T}\bot$ is the initial type $\mathbf{0}$, it clearly suffices to define $\equiv$ by

$$a \equiv b \quad := \quad \mathbf{T}a \longleftrightarrow \mathbf{T}b.$$

- $A = \mathbf{T}c, B = \mathbf{T}d$. Then c and d are of type $\mathbf{2}$.

$$a \equiv b \quad := \quad c \equiv d.$$

- $A = \forall F, B = \forall G$, where F and G have bases C and D, respectively. Then C and D are congruent and F and G are congruent.

$$a \equiv b \quad := \quad \forall x{:}C \exists y{:}D(x \equiv y) \wedge \forall y{:}D \exists x{:}C(x \equiv y)$$

$$\wedge \forall x{:}C \forall y{:}D(x \equiv y \longrightarrow ax \equiv by).$$

- $A = \exists F, B = \exists G$ where F and G have bases C and D, respectively. Again, C and F are respectively congruent to D and G.

$$a \equiv b \quad := \quad a1 \equiv b1 \wedge a2 \equiv b2.$$

It is easy to deduce that

$$\top \not\equiv \bot$$

$$\forall x{:}\mathbf{2}\ [x \equiv \top \ \vee \ x \equiv \bot]$$

$$\forall x{:}\mathbf{2} \forall y{:}\mathbf{T}x \forall z{:}\mathbf{T}xy \equiv z$$

and that extensional equality is transitive and symmetric. I.e.

$$\forall x{:}A\ \forall y{:}A\ \forall z{:}A\ [x \equiv y \ \wedge \ y \equiv z \quad \longrightarrow \ x \equiv z]$$

$$\forall x{:}A \forall y{:}A[x \equiv y \ \longrightarrow \ y \equiv x].$$

Moreover, for a given object a of type A *in* Σ, there is a deduction of $a \equiv a$. But, alas, we cannot deduce for an arbitrary type A that extensional equality on A is reflexive:

$$\forall x{:}A\ [x \equiv x]. \tag{3}$$

The problem case is, of course, when A is of the form $\forall F$, where F has some base B. It would be consistent to add non-extensional functions to Σ.

On the other hand, we could consider introducing the principle (3) as a 'regulative principle'. Thus, for each object b of a type A that is introduced, there must be a proof E'_b provided of $b \equiv b$. Then we would introduce the constant

$$E(A) : \forall x : A.(x \equiv x).$$

by means of the definition

$$E(A)b := E_b.$$

However, we are not done, since according to our requirement for introducing objects, we have to prove the reflexivity of $E(A)$, itself: $E(A) \equiv E(A)$. I.e., we have to have a proof of

$$\forall x : A \forall y : A[x \equiv y \longrightarrow E(A)x \equiv E(A)y]$$

The proof of this turns out to be something of a *tour de force*.

Proposition. In any polynomial extension on Σ, from $p\colon\ a \equiv a', q\colon\ b \equiv b'$ and $r\colon\ a \equiv b$, we can construct a proof of $p \equiv q$.

In particular, then, from proofs $r : u \equiv v$ we can construct a proof of $E(A)u \equiv E(A)v$. We prove the theorem by induction on the type of a. Note that from p, q and r we obtain a proof s of $a' \equiv b'$.

CASE 1. a is of type $\mathbf{2}$. Then

$$p\colon\ \mathbf{T}a \longleftrightarrow \mathbf{T}a' \qquad q\colon\ \mathbf{T}b \longleftrightarrow \mathbf{T}b'$$

Thus

$$p1\colon\ \mathbf{T}a \longrightarrow \mathbf{T}a' \qquad q1\colon\ \mathbf{T}b \longrightarrow \mathbf{T}b'$$

We need $p1 \equiv q1$, i.e.

$$\forall x : \mathbf{T}a\ \exists y : \mathbf{T}b(x \equiv y) \qquad \forall y : Tb\ \exists x : \mathbf{T}a(x \equiv y)$$

and

$$\forall x\colon\ \mathbf{T}a\ \forall y\colon\ \mathbf{T}b[x \equiv y \longrightarrow p1x \equiv q1y]$$

$r1 : \mathbf{T}a \longrightarrow \mathbf{T}b$ and $r2 : \mathbf{T}b \longrightarrow \mathbf{T}a$. So, if $u : \mathbf{T}a$ and $v : \mathbf{T}b$, then $r1u : \mathbf{T}b$ and $r2v : \mathbf{T}a$. So the first two conditions are satisfied. The last condition is just $(\mathbf{T}a \longleftrightarrow \mathbf{T}b) \longrightarrow (\mathbf{T}a' \longleftrightarrow \mathbf{T}b')$, which is obtained from p and q. By symmetry, we also have $p2 \equiv q2$ and so $p \equiv q$.

CASE 2. $a : \mathbf{T}c$. Then $a' : \mathbf{T}c'$, $b : \mathbf{T}d$, $b' : \mathbf{T}d'$. Then $a \equiv a'$ is just $c \equiv c'$, etc.; and this case reduces to Case 1.

CASE 3. $a : \exists x : AF(x)$. Then $a1 : A$ and $a2 : F(a1)$, $p1 : a1 \equiv a'1, q1 : b1 \equiv b'1$, etc; and so, by the induction hypothesis, $p1 \equiv q1$ and $p2 \equiv q2$.

CASE 4. $a : (\forall x : A.F(x)), b : (\forall y : B.G(y)), a' : (\forall x' : A'.F'(x')), b' : (\forall y' : B'.G'(y'))$. In the following, I will drop the A in $x : A$, and similarly for x', y and y'. So p, as a proof of

$$\forall x \exists x' (x \equiv x') \wedge \forall x' \exists x (x \equiv x') \wedge \forall x x' [x \equiv x' \longrightarrow ax \equiv a'x']$$

has three components, p^0, p^1, p^2 which are proofs of the conjuncts, respectively, and similarly for q, r and s. We need to show for $i \equiv 0, 1, 2$ that $p^i \equiv q^i$. To prove $p^0 \equiv q^0$, we need

$$\forall x \exists y (x \equiv y) \qquad \forall y \exists x (x \equiv y)$$

which have the proofs r^0 and r^1, and

$$\forall xy [x \equiv y \longrightarrow p^0 x \equiv q^0 y]$$

Let $u : A, v : B$ and assume $u \equiv v$. $p^0u1 : A'$, $q^0v1 : B'$, $p^0u2 : u \equiv p^0u1$ and $q^0v2 : v \equiv q^0v1$. So $p^0u1 \equiv q^0v1$ and hence, by the induction hypothesis, $p^0u2 \equiv q^0v2$. Thus, $p^0 \equiv q^0$.

By symmetry, $p^1 \equiv q^1$.

As for p^2 and q^2, we have

$$p^2 : \forall x x' [x \equiv x' \longrightarrow ax \equiv a'x'] \qquad q^2 : \forall y y' [y \equiv y' \longrightarrow by \equiv b'y']$$

$p^2 \equiv q^2$ is clearly equivalent to the conjunction of

$$\forall x \exists y (x \equiv y) \qquad \forall y \exists x (x \equiv y)$$

$$\forall x' \exists y' (x' \equiv y') \qquad \forall y' \exists x' (x' \equiv y')$$

which have the proofs r^0 and s^0, and

(4) $$\forall xx'yy'[x \equiv y \wedge x' \equiv y' \longrightarrow p^2xx' \equiv q^2yy']$$

Let u, u', u^* be of types A, A' and $u \equiv u'$, respectively, and let v, v' and v^* be of types B, B' and $v \equiv v'$, respectively. Then $p^2uu'u^* :\ au \equiv a'u'$ and $q^2vv'v^* :\ bv \equiv bv'$. Let $w : u \equiv v$ and $w' : u' \equiv v'$. Then $r^2uvw : au \equiv bv$. So by the induction hypotheses, $p^2uu'u^* \equiv q^2vv'v^*$ follows from the assumptions w and w'. This demonstrates 4.

References

Bernays, P. [1959]. Über eine naturliche Erweiterung des Relationkalkuls, *[Heyting 1959]* pp. 1–14.

Curry, H. Feys, R. [1958]. Combinatory logic i, studies in logic and the foundations of mathematics. 2nd edition 1968.

Heyting, A. (ed.) [1959]. *Constructivity in Mathematics*, Amsterdam: North-Holland.

Howard, W. [1980]. The formula-as-types notion of construction, pp. 479–490.

Martin-Löf, P. [1998]. An intuitionistic theory of types.

Quine, W. [1951]. *Mathematical Logic: Revised Edition*, Cambridge: Harvard University press.

Quine, W. [1960a]. Variables explained away, *Proceedings of the Americal Philosophical Society* . Reprinted in [Quine 1966a, 227–235].

Quine, W. [1966a]. *Selected Logic Papers*, New York: Random House.

Sambin, G. Smith, J. (eds) [1998]. *Twenty-Five Years of Constructive Type Theory*, Oxford: Oxford University Press.

Schönfinkel, M. [1924]. Über die Bausteine der mathematischen Logik, *Mathematische Annalen* **92**: 305–316.

Tait, W. [1996]. Extensional equality in classical type theory, *in* W. DePauli-Schimanovich, E. Köhler F. Stadler (eds), *The Foundational Debate: Complexity and Constructivity in Mathematics and Physics*, pp. 219–234.

Tait, W. [1998b]. Variable-free formalization of the Curry-Howard type theory, *[Sambin Smith 1998, 265–274]* .