Towards a computation system based on set theory

Article *in* Theoretical Computer Science · September 1988

DOI: 10.1016/0304-3975(88)90115-6 · Source: dx.doi.org

1 author:

Michael Beeson
San Jose State University
100 PUBLICATIONS 1,231 CITATIONS

Theoretical Computer Science 60 (1988) 297-340
North-Holland

TOWARDS A COMPUTATION SYSTEM BASED ON SET THEORY

Michael J. BEESON

Department of Mathematics and Computer Science, San Jose State University, San Jose, CA 95192, U.S.A.

Communicated by C. Böhm
Received March 1986
Revised December 1987

Abstract. An axiomatic theory of sets and rules is formulated, which permits the use of sets as data structures and allows rules to operate on rules, numbers, or sets. We might call it a "polymorphic set theory". Our theory combines the λ-calculus with traditional set theories. A natural set-theoretic model of the theory is constructed, establishing the consistency of the theory and bounding its proof-theoretic strength, and giving in a sense its denotational semantics. Another model, a natural recursion-theoretic model, is constructed, in which only recursive operations from integers to integers are represented, even though the logic can be classical. Some related philosophical considerations on the notions of set, type, and data structure are given in an appendix.

Contents

1. Set theory and computer science

The evolution of programming languages has in part consisted of the search for ever more flexible and general data structures. This problem was confronted in the nineteenth century by mathematics; the history of the development of mathematics

can be seen as an elaboration of data structures. First one had the integers, then the rationals, the algebraic numbers, after some struggle the real and complex numbers, the general concept of function, etc. At last it was realized that set theory could serve as a universal language, in which all the necessary data structures could be constructed from the primitive data structure, set. Perhaps computer science could take advantage of this experience. This paper can be viewed as a preliminary investigation into what a programming language based on set theory might be like.

On the other hand, in the last half-century mathematicians have defined functions by identifying them with the sets which are their graphs. Rigor is identified with set theory, so the definition of a function as a rule has to go. This is particularly inappropriate for computer science, and even many mathematicians are more interested than before in rules for computing the solutions of problems. We shall resurrect functions-as-rules and give them the place in set theory that they deserve. Thus this paper can also be viewed as an investigation in the foundations of mathematics.

In order to be manipulated by a computer, a set must be represented by a concrete object, ultimately by a sequence of zeroes and ones. That does not mean, however, that only finite sets can be manipulated. For example, the data type *bignum* of nonnegative integers of arbitrary size is common; we shall denote it more simply by N. This is an example of representing an infinite set by a (finite) name, in this case together with a defining property enabling us to compute membership in the set.

No modern programming language in common use is adequate to the demands of algorithmic mathematics. Knuth [32], making the same point, uses "pidgin Pascal" to express an algorithm extracted from Bishop's constructive proof of the Weierstrass approximation theorem that any continuous function can be approximated by a polynomial. That algorithm takes an argument of type f: *real* → *real*, another argument of the same type for the modulus of continuity of f, a third argument ε of type *real* for the desired tolerance of approximation, and outputs a polynomial, which can be coded by a finite sequence of reals (of unknown length). This example illustrates the need for more general typing facilities. Although many efforts are proceeding in this direction (some of which are discussed in the next section), the simple approach of adding rules to set theory seems not to have been explored.

Set theory should be a polymorphic language; that means that data types can be arguments of rules, as well as being used to specify domains and ranges.[1] Here a difficulty arises with the standard set theories: consider the operation of union. Union is a perfectly good operation on two sets; but how can the operation "union" itself be represented in set theory? Its graph is a proper class, so we cannot use the usual device of identifying a function with the set which is its graph.

Closely related to the problem of polymorphism is the problem of type embedding. In mathematics, 5 is an integer, a rational number, a real number, and a constant

[1] Sometimes *polymorphic* refers to a procedure that accepts arguments of several different types. This can be viewed as a single procedure, one of whose arguments is a type.

real-valued function. In most typed programming languages, one has to have a different "5" of each of these types, and even a different "5" of type *int* and type *bignum*. In advanced computation systems such as *Algos* [22], considerable effort is expended to make the system automatically convert between these types when necessary. This problem would have to be solved by any implementation of set theory as a computation system, and the theoretical explorations in this paper may cast some light on the situation.[2] In order to achieve a polymorphic set theory, we take both *rule* and *set* as primitive, instead of only *set* as in mathematics. Our idea is essentially this: write down the axioms of λ-calculus for rules, and the axioms of Zermelo–Fraenkel set theory, extended to this language.

This is such an obvious idea that one wonders why it has not been done before. Here is a conjecture: a theory of sets and rules could not have been created by Zermelo in 1908, because the necessary understanding of the nature of algorithms had not yet been achieved. That had to await the work of Church, Turing, Kleene, and Gödel in the 1930s. By that time, however, theory of sets was so well-established that nobody thought of combining it with a theory of rules. Moreover, until recently, set theory was perceived as inherently nonconstructive, and there was a psychological block to combining set theory with algorithms or constructive mathematics. Neither of these obstacles applies today: the time is ripe.

2. A paradox

It came as a surprise that, when one straightforwardly carries out the program of combining λ-calculus with set theory, the resulting theory is inconsistent. This section will present the paradox, which depends on three intuitively appealing properties: extensionality, separation, and recursion. These three properties will be discussed first.

2.1. Extensionality. This principle says that two sets are equal if and only if they have the same members. This may be regarded as the definition of equality for sets. While a given set may have many different symbolic representations which can be manipulated by a computer (or none at all), these representations are not the same as the set. A property of representations which does not respect the relation of extensional equality (having the same members) is not a set-theoretic property; it is not a property of sets, but of their representations.

2.2. Separation. If ϕ is a "definite property", and a is a set, then $\{x \in a : \phi(x)\}$ is a set. Here the words "definite property" are due to Zermelo [40], who first wrote

[2] The claim that the problems of polymorphism and type embedding are closely related can be explained by the following example. Consider the operation $swap(x, y) = (y, x)$. This should should be implemented by a polymorphic procedure independent of the types of x and y. Then, for each particular choice of types of x and y, we get a particular *swap* operation; the types of these operations should be embeddable in each other. In other words, a polymorphic procedure may have many types.

down an axiomatic set theory simlar to that in use today. The operation by which this set is produced from a is a first-rate example of a legitimate operation on sets; there should be an operation c_ϕ such that $c_\phi(a) = \{x \in a : \phi(x)\}$.

2.3. Recursion. The operators we have been discussing are in general partially defined, and should satisfy the recursion theorem, according to which for each operation g there is an operation f such $f(x) \simeq g(x, f)$.

But as it turns out, these three principles taken together are contradictory:

2.4. Theorem (Gordeev). *Extensionality, the recursion theorem, and the existence of an operation c_ϕ corresponding to separation are contradictory.*

Proof. Let $g(z, f) = \{x \in \{\emptyset\} : f(z) = x\}$, by separation, where $\emptyset$ is the empty set. Introduce f by the recursion theorem so that $f(z) = g(z, f)$. Since g is total, so is f; but then $f(f)$ is a set, namely $f(f) = \{x \in \{\emptyset\} : f(f) = x\}$. If $f(f) = \emptyset$, then $\emptyset \in f(f)$, contradiction, so $f(f) \neq \emptyset$. But if $x \in f(f)$, then $x = \emptyset$ and $x = f(f)$, contradiction. Hence $f(f)$ has no members. By extensionality then $f(f) = \emptyset$, contradiction. □

This theorem was first discovered in connection with theories of sets and rules developed by Feferman [14], in which extensionality is not an axiom. It was therefore not regarded as a paradox, but simply as a refutation of extensionality in the context of those theories. In the present context, however, Gordeev's theorem does seem paradoxical at first, and it prompts a closer examination of the foundational viewpoint which must underlie the theories we intend to construct.

3. Rules and sets: reflections on the foundations of mathematics

In mathematics we find two kinds of "mathematical entities": the concrete and the abstract. By concrete entities we mean objects which can be completely represented by a finite sequence of symbols, capable of being stored in a computer (although no fixed bound on the size of concrete objects is imagined). By an abstract entity, on the other hand, we mean an object which by its nature is infinite and can never be completely displayed. An abstract entity may however be represented in various ways; the example N has already been discussed.

A discovery of the early axiomatizers of set theory was that concrete entities can be represented or "coded" by suitable sets, so that from a formal point of view we may pretend that "everything is a set". The resulting axiomatizations have an appealing simplicity, and no doubt it was partly due to this simplicity that set theory came to be regarded as a "universal language" for mathematics, a status which it enjoys to this day. Only certain dissenters from the mainstream of mathematics have resisted the doctrine that "everything is a set". We have in mind various schools of

constructive thought, such as the intuitionists (followers of Brouwer); the Russian constructivists (following Markov), who reject all abstract objects out of hand and work only with symbolic representations; and more recently those working in the style of Bishop [5], who accepted a notion of sets as abstract objects.

All of these schools regard the notion of *rule* or *law* as fundamental. Recall that, for Brouwer, "eine Menge ist ein Gesetz" (a set is a law). It is also interesting that if we go back to the period before Zermelo's axiomatization, the concept of "law" plays a role. For example, the notion of "transformation" (Abbildung) plays an important role in Dedekind's *Was sind uns was sollen die Zahlen*: he takes "transformations" as laws that transform elements s of a set S to "determinate things" $\phi(s)$. He then anticipates Fraenkel's principle of "replacement" by stating that the image $\phi(S)$ of a set S is again a set.

The distinction made above between abstract and concrete entities opens the way for another important distinction, that between *rule* and *algorithm*. An algorithm operates on concrete objects and produces (if anything) concrete objects. A rule, on the other hand, may operate on and produce either concrete or abstract objects. For example, there is an operation $\mathbf{u}$ such that $\mathbf{u}(x)$ is the set known as the union of x, usually written $\bigcup x$, whose members are the members of the members of x. The rule $\mathbf{u}$ is not representable in set theory in its usual formulation: it is not a set, nor is its graph a set. It is a rule. Another example is the powerset operation $\mathcal{P}$.

One thesis of this paper is that rules as fundamental to mathematics and computer science as sets are. For example, the concept of "function" has its roots in the idea that "a function is a rule". The gradual development of that idea into the definition of a function as a single-valued set took a long time (see [36]), and perhaps ought to be reconsidered for the foundations of computer science. Even in mathematics the idea of "function-as-a-rule" retains a certain appeal, and the definition of "function-as-a-set" is usually justified when first presented by reference to the intuitive concept of a rule, which shows that the concept of a rule is actually more fundamental.

Rules are supposed to be concrete objects, given by symbolic representations. A question now arises: how does a rule operate on a set, which is an abstract object? The operation $\mathbf{u}$, for example, is supposed to operate on *any* set, not just on sets which happen to have symbolic representations. There are various algorithms corresponding to $\mathbf{u}$, which enable us to compute symbolic representations of $\bigcup x$ from specified kinds of symbolic representations of x, but these algorithms are not to be identified with the opeation $\mathbf{u}$ itself. We shall have to take the application operation as primitive. When we give a rule, we must tell how to apply it, i.e., what the answer is at a given argument. We may also define algorithms operating on symbolic representations of sets. It may happen that such an algorithm is extensional, in the sense that it takes all representations of any given set x onto representations of the same set. In that case, the algorithm can be counted as a rule, although it would be defined only in those sets that have a symbolic representation of the kind involved in the definition of the algorithm. There may also be algorithms which are not

extensional; for example, if sets are represented by listing their members, then the function which picks the first element of a list will not be extensional.

With these distinctions in hand, we can "resolve" the paradox posed by Gordeev's theorem. The proof constitutes the construction of a particular algorithm which is demonstrably nonextensional. The author does not consider this "resolution" any more definitive than the so-called resolutions of the paradoxes of classical set theory. We shall proceed by retreating to axioms which block the contradiction, yet are useful for practical purposes.

The main idea is to restrict the separation axiom to formulae ϕ of ordinary set theory, i.e., formulae which do not involve the application relation. This turns out not to be very restrictive; in practice, we can often define sets that we need (including the dependence on parameters), even when the definition appears to require separation for a formula involving the application of operations. The method is a process of substitution. For example, suppose we want to define $g(x, b) = \{u: \exists x \in b(u \in f(x))\}$. There will be a certain operation c_ϕ such that $c_\phi(a, w) = \{x \in a: x \in w\}$. Then $g(x, b) = c_\phi(\mathbf{u}(\mathbf{im}(b, f)), f(x))$, where as above $\mathbf{u}$ is union and $\mathbf{im}(b, f)$ is the image of b under f.

It is instructive to see why this method does not permit us to define the set used in Gordeev's proof. Try $h(z, y) = \{x \in \{\emptyset\}: y = x\}$ and $g(z, f) = h(z, f(z))$. But now, although h is total, there is no reason why g is total, and the argument in Gordeev's proof only shows that $f(f)$ is undefined.

To avoid any confusion, one point should be reiterated. Gordeev's theorem does *not* show the inconsistency of extensionality, the recursion theorem, and separation. Rather, it shows the inconsistency of extensionality, the recursion theorem, and the dependence on parameters *by a rule* of sets defined by separation.

4. Related work

Zermelo [40] formulated axioms for set theory which form the basis of modern set theory. Zermelo said that he was motivated by "set theory as it is historically given". That is, he attempted to write down axioms which accounted for mathematical practice, and which he believed to be free of paradoxes. Let us call this the "pragmatic" approach to axiomatization, as opposed to the "philosophical" approach in which one attempts to justify one's axioms. The philosophical approach was taken by Cantor and Dedekind (see the Appendix of this paper), but Zermelo's main motivation was to convince people that he had really proved that the reals could be well-ordered, which accounts for his pragmatism.

Von Neumann [38] showed that one could take the concept "function" as primitive instead of "set", and define "set" in terms of "function". This was, however, a theory of functions, not a theory of rules; no computability was implied.

Bishop [5] viewed mathematics as a "high-level programming language". In his book, he made no attempt to formalize that programming language, although in

unpublished work he did so. His formalization remained unpublished, because his theory contained a paradox.

Martin-Löf [33, 34] introduced "type theories", in which one forms types which are similar to (but not as general as) sets. In the most recent version of these theories, Martin-Löf has explicitly forbidden the formation of types of types, but allows only the formation of types of *names of types*, espousing the principle that one may form types only of concrete objects. Martin-Löf's approach is philosophical as opposed to pragmatic; he views his formal theories as an incomplete description. He clearly believes he has replaced Cantor's definition of a set with one which (if not equally simple) is at least correct, and hence does not lead to paradoxes. (It is interesting to note, however, that the earliest version of his theory was *also* inconsistent.) A group in Göteborg has implemented a system based on Martin-Löf's theories on the computer; it is called "Göteborg LCF".

Feferman [14, 15] introduced theories of operations and classes. These are based on a philosophy that the "universe" V consists of objects representable by finite symbolic expressions. One may form sets or "classes" of these objects. The objects include names or representations of the classes. If we understand Feferman's viewpoint correctly, one is never forming sets of abstract entities; for example, one never forms sets of sets. Instead, one forms sets of concrete representations of sets. This viewpoint justifies Feferman's "elementary comprehension axiom" that permits the formation of a universe V of all objects. The Russell paradox is blocked by the syntactic restriction to "elementary" formulae in the comprehension axiom. Hayashi [27] has implemented a version of Feferman's theory in LISP, in a program called PX. His theory replaces the combinators in Feferman's theory by the primitives of LISP.

Myhill [42] and Friedman [19, 20] introduced theories in the language of classical set theory, but with intuitionistic logic. These theories, because of their intuitionistic logic, are indirectly connected to rules.

Graves [22] designed and implemented a theory of types based on category theory, more specifically on topos theory. Gordon, Milner, and Wadsworth [24] describe a system with rather general typing facilities known as "Edinburgh LCF", for which a denotational semantics based on the Scott-Strachey formalism has been given. Constable [9] has (with others) developed a system called NuPrl (two of whose predecessors are described in [10, 11]). These systems include proof-checkers operating in a typed, functional environment, whose theoretical basis belongs to the lineage of Martin-Löf's theories.

Belonging to the same lineage are at least three typed, fully functional programming languages. (This phrase means that procedures are data objects with types.) These are ML, Pebble, and QJ. ML was originally developed as a "metalanguage" for LCF, which accounts for its name; its structure makes it appropriate that these are also Martin-Löf's initials. Pebble, which is being developed at DEC by Burstall and Lampson [8], permits types as values and allows some rather general type constructions; in particular bindings of variables are ordinary data objects. QJ was

created by Sato as a uniform environment for the specification, execution, and verification of programs. The system *Algos* mentioned above is similar, but its lineage is category theory.

After this summary, we shall attempt to place the present work in context. There were three lines of development of theoretical, logical systems for constructive mathematics in the seventies: Martin-Löf's type theories, Feferman's not-strictly-typed theories, and Friedman-Myhill's constructive set theories. Each of these is capable of being implemented as a programming language with sufficient power to formalize mathematics in an interactive proof-checking environment. This line of research is well-developed for Martin-Löf's type theories, being developed for Feferman's theories by Hayashi, and so far untouched for the constructive set theories. As they stand, the constructive set theories are too crude since they do not allow a direct treatment of functional application. In this paper we correct this defect, creating constructive set theories suitable for computerization.

5. Preliminary description of ZFR

We shall formulate a theory **ZFR**; the letters stand for "Zermelo-Fraenkel set theory with Rules". This section describes the theory; the next section gives a list of its axioms for reference.

In formulating theories of sets and rules, the first matter requiring attention is the choice of language. In classical Zermelo-Fraenkel set theory **ZF**, the assumption is made at the outset that every mathematical object is a set. This assumption is actually false: the integers are not sets (New Math to the contrary). Formally speaking, the Von Neumann integers—that is, the sets generated from the empty by the operation $x \cup \{x\}$—are isomorphic to the integers, and can serve as representatives of the integers. But to see that this is harmless, we must work outside the conceptual framework of **ZF**. Our present object is to formulate theories which more naturally reflect our intuitive views about the foundations of mathematics; hence, we do not wish to begin by assuming everything is a set. Moreover, we wish to leave the way open for computer-oriented versions of the theory in which there may be other "atomic" data types such as, for example, the symbolic atoms of LISP.

We therefore include unary predicates "$S(x)$" and "$N(x)$" for "x is a set" and "x is a nonnegative integer" respectively. It will not be necessary to include a predicate for "x is a rule". We shall not put into our theory any axioms restricting the kinds of objects there are, such as "everything is either an integer or a set". As a matter of convenience, we shall use the letters a, b, u, v, w (with or without subscripts) for sets, and i, j, k, n, m for integers. By this we mean that variables denoted by these metavariables are implicitly restricted to the unary predicates S and N. There will be a binary relation "$\in$" for membership.

We shall need a means of speaking of the application of a rule to an object. There are two ways to proceed:

(i) We can introduce a 3-ary relation **App**(f, x, y) for: the result of applying f to x is y.

(ii) We can introduce a function symbol **Ap** so that **Ap**(f, x) is the result of applying f to x.

Approach (ii) seems more natural, but it necessitates a reformulation of logic since **Ap**(f, x) will sometimes be undefined. One suitable reformulation of logic has been given in [3], where it is called **LPT** (logic of partial terms). There it is explained how to interpret a theory formulated in **LPT** into a theory formulated in style (i) above, with a 3-ary **App** relation. We shall formulate our theories in **LPT**. In practice, we shall write fx, $f(x)$, or (fx) instead of **Ap**(f, x), We follow the conventions of combinatory logic: xyz abbreviates $(xy)z$ (association to the left), and $f(x, y)$ abbreviates fxy.

Logic of partial terms

The following description of **LPT** is reproduced from [3], where further details may be found. **LPT** is a logic in the same sense as the predicate calculus. If we are given any collection of predicate symbols, function symbols, and constants as in the usual predicate calculus, there will be a language in **LPT** based on these symbols. The rules for forming terms are the same as in ordinary predicate calculus. Every atomic formula in the usual sense is still an atomic formula; but there is one more kind of atomic formula, namely: if t is a term, then $t\!\downarrow$ is an atomic formula. This may be read "t is defined". It should be emphasized, however, that the intended meaning is that the term "t" denotes something. That is, one says of an object that it exists, of a term that it denotes or is defined. All objects exist, of course, so to say that something does not exist is a figure of speech; what is meant is that the term one has mentioned does not denote.

In case equality is part of the language, we use $t \cong s$ to abbreviate $(t\!\downarrow \to t = s) \wedge (s\!\downarrow \to t = s)$. In words: if either t or s denotes anything, then they both denote the same thing. Note, however, that $\cong$ is not an official part of the language.

We shall use the notation $A[t/x]$ to mean the result of substituting t for the free occurrences of x in A. The customary inference from $\forall x A$ to $A[t/x]$ is not valid if t is a nondenoting term: "if everything exists, then the king of France exists" is an invalid inference since the antecedent is true but the consequent is false. We are now ready to set out a list of rules and axioms for making correct inferences in LPT. In this list, t and s are terms, while x and y are variables.

Axioms and rules of LPT

$$\frac{B \to A}{B \to \forall x A} \quad \text{if } x \text{ is not free in } B, \tag{Q1}$$

$$\frac{A \to B}{\exists x A \to B} \quad \text{if } x \text{ is not free in } B, \tag{Q2}$$

$$\forall x A \wedge t\downarrow \to A[t/s], \tag{Q3}$$

$$A[t/x] \wedge t\downarrow \to \exists x A; \tag{Q4}$$

$$x = x \wedge (x = y \to y = x), \tag{E1}$$

$$t \cong s \wedge \phi(t) \to \phi(s), \tag{E2}$$

$$t = s \to t\downarrow \wedge s\downarrow; \tag{E3}$$

$$R(t_1, \ldots, t_n) \to t_1\downarrow \wedge \cdots \wedge t_n\downarrow, \tag{S1}$$

$$c\downarrow \quad \text{for constants } c, \tag{S2}$$

$$x\downarrow \quad \text{for variables } x. \tag{S3}$$

Note that E3 is a special case of S1. Another special case of S1 worthy of mention is

$$f(t_1, .., t_n)\downarrow \to t_1 \wedge \cdots \wedge t_n\downarrow.$$

With a suitable logic at hand, we are now in a position to set down some axioms about rules. Before doing this, we have to choose a suitable language in which to express our rules. The proliferation of modern programming languages illustrates the range of possibilities here. Since our present purpose is primarily theoretical, as opposed to being aimed at a practical implementation, it will be wise to choose the simplest possible language, even if that makes it difficult in practice to write down specific rules. We therefore choose the language of combinatory logic, which is the simplest programming language in the world. Our theory will include two constants $\mathbf{k}$ and $\mathbf{s}$, with the axioms

$$\mathbf{k}xy = x, \qquad \mathbf{s}xyz \cong xz(yz).$$

These axioms permit the proof of the recursion theorem, according to which we can find, given g, an f such that $fx \cong g(x, f)$. In addition, they permit the construction of λ-terms: if t is any term, then there is another term $\lambda x.t$ such that $(\lambda x.t)x \cong t$ (where x is a variable). Note that this is not a term-formation rule but a theorem. See [2, 12, 15] for details of these results.

The remaining axioms about rules concern the existence of a certain specific rules. First of all, there is a constant $\mathbf{s}_N$ for the successor function on the integers. Second, there are several axioms corresponding to the set-theoretic axioms of pairing, separation, union, and powerset, which introduce constants as follows; $\mathbf{p}(x, y)$ is the unordered pair of x and y; $\mathbf{u}x$ is the union of x; $\mathbf{P}x$ is the powerset of x, and

$$c_\phi(a, y_1, \ldots, y_n) = \{x \in a : \phi(x, y_1, \ldots, y_n)\}.$$

The formulation of the replacement axiom in our theory is simply that the image of a set a under an operation f such that $f(x)$ is defined for all $x \in a$ is a set $\mathbf{im}(a, f)$. The rule $\mathbf{im}$ is itself a mathematical object. This is evidently what we try to express

in much less natural language in the usual formulation of the replacement axiom,

(Rep) $\forall x \in a \exists! y \phi(x, y) \rightarrow \exists b \forall x \in a \exists y \in b \phi(x, y)$.

Moreover, this version of the axiom is not self-evident; the version about **im** is the self-evident version. **Rep** can be derived from the axiom about **im** using a suitable version of the axiom of choice, but then one has to justify *that* if one can.

The other two set-theoretical axioms, extensionality and $\in$-induction, are not about rules at all, but about the nature of sets.

The constants c_ϕ are included only for formulae ϕ which do not contain **Ap** and do not contain any constants c_ψ; this restriction will enable us to block the paradox discussed above.

6. Formal specification of ZFR

It is natural to use intuitionistic logic when contemplating a computation system; but **ZFR** makes sense with or without the law of the excluded middle. To fix the notation, we shall use **ZFR** for the version with classical (ordinary) logic, and **IZFR** for the version with intuitionistic logic. Since **ZFR** can be obtained from **IZFR** by adding the law of the excluded middle, we specify **IZFR** below.

Language of IZFR. Unary predicate symbols S and N for sets and numbers. Binary relation symbols $\in$ and $=$. Function symbol **Ap**. Constants λ, **k**, **s**, $\mathbf{s_N}$, 0, $\emptyset$, $\mathscr{P}$, **d**, **p**, **N**, **im**, $\mathbf{c}_\phi$ for each primitive formula ϕ; the concept "primitive formula" is defined next.

Primitive formula. One not containing **Ap** or any constant $\mathbf{c}_\phi$.

Logic of IZFR. **LPT** as specified above, with intuitionistic logic and equality axioms.

Axioms of IZFR

(A1) *Extensionality*: $\forall x(x \in a \leftrightarrow x \in b) \rightarrow a = b$.

(A2) *Pairing*: $S(\mathbf{p}yz) \wedge \forall x(x \in \mathbf{p}yz \leftrightarrow x = y \vee x = z)$.

(A3) *Union*: $S(\mathbf{u}a) \wedge \forall x(x \in \mathbf{u}a \leftrightarrow \exists u(u \in a \wedge x \in u))$.

(A4) *Empty set*: $S(\emptyset) \wedge \forall x(x \notin \emptyset)$.

(A5) *Infinity*: $S(\mathbf{N}) \wedge \forall x(x \in \mathbf{N} \leftrightarrow N(x))$.

(A6) *Separation*: $S(\mathbf{c}_\phi(a, y_1, \ldots, y_n))$
$\wedge \forall x(x \in \mathbf{c}_\phi(a, y_1, \ldots, y_n) \leftrightarrow x \in a \wedge \phi(x, y_1, \ldots, y_n))$.

(A7) *Images*: $\forall x \in a(fx\downarrow) \rightarrow S(\mathbf{im}(a, f))$
$\wedge \forall z(z \in \mathbf{im}(a, f) \leftrightarrow \exists x \in a(fx = z))$.

(A8) *Powerset*: $S(\mathscr{P}a) \wedge \forall x(x \in \mathscr{P}a \leftrightarrow S(x) \wedge \forall z \in x(z \in a))$.

(A9) $\in$*-induction*: $\forall u((\forall x \in u \phi(x)) \rightarrow \phi(u)) \rightarrow \forall u \phi(u)$.

(B1) $z \in x \to S(x)$.
(B2) *Cases*: $\mathbf{d}nnxy = x \wedge (n \neq m \to \mathbf{d}nmxy = y)$.
(B3) *Successor*: $N(0) \wedge N(\mathbf{s}_N n) \wedge (\mathbf{s}_N n = \mathbf{s}_N m \to n = m) \wedge \mathbf{s}_N n \neq 0$.
(B4) *Induction*: $\phi(0) \wedge \forall n(\phi(n) \to \phi(\mathbf{s}_N n)) \to \forall n \phi(n)$, all formulae ϕ.

Conventions. a, b, u, v, w are (meta)variables for sets; i, j, k, n, m are for numbers. These conventions have been used to abbreviate the axioms. Other variables such as x, y, z are unrestricted.

Remark. Note that the pairing axiom permits us to form a set $\mathbf{p}xy$, usually written $\{x, y\}$, from any two objects x and y, not only from two sets.

7. Axioms of set theory

In this section we list for reference the axioms of Zermelo–Fraenkel set theory. These axioms are formulated in a language with binary relation symbols $\in$ and $=$, and no other constants, relation, or function symbols.

Extensionality: $\forall x(x \in a \leftrightarrow x \in b) \to a = b$.
Pairing: $\exists a(x \in a \wedge y \in a)$.
Union: $\exists a \forall x(x \in a \leftrightarrow \exists b \in a(x \in b))$.
Separation: $\exists a \forall x(x \in a \leftrightarrow x \in b \wedge \phi)$ (a not free in ϕ).
Infinity: $\exists a(\exists x \in a \wedge \forall x \in a \exists y \in a(x \in y))$.
Powerset: $\exists a(\forall z(z \in x \to z \in b) \to x \in a)$.
$\in$-*induction*: $\forall u(\forall x \in u \phi(x) \to \phi(u)) \to \forall u \phi(u)$.
Empty set: $\exists a \forall x(x \notin a)$.
Replacement: $\forall x \in a \exists! y \phi \to \exists b \forall x \in a \exists y \in b \phi$.
Collection: $\forall x \in a \exists y \phi \to \exists b \forall x \in a \exists y \in b \phi$.

These are the versions of the axioms which Friedman [18] discovered were suitable for use with intuitionistic logic; note that they are the same as the usual axioms except that the axiom of foundation has been replaced by $\in$-induction. Also, the axioms of replacement and collection are equivalent if classical logic is allowed, but not with intuitionistic logic [21]. We therefore have the following set theories:

- **IZF-Rep**: all of the above axioms except collection, with intuitionistic logic,
- **IZF-Col**: all of the above axioms, with intuitionistic logic,
- **ZF**: all of the above axioms, with classical logic.

Of course, collection implies replacement, so there is no need to include replacement on the list of axioms of **ZF** or **IZF-Col**.

This paper can be understood without knowing anything about intuitionistic logic, but it is intuitionistic logic that would be built into a computation system based on **IZFR**, and so a background in intuitionistic set theory is relevant. See [4, Chapter VIII].

8. Functions versus operations

By a "function", or sometimes for emphasis a "set-theoretic function", we mean as usual a single-valued set:

$$\mathit{Funct}(f) \leftrightarrow S(f) \wedge \forall z \in f \exists x \exists y (z = \langle x, y \rangle)$$
$$\wedge \forall x \forall y \forall z (\langle x, y \rangle \in f \wedge \langle x, z \rangle \in f \rightarrow y = z).$$

Note that although ordered pairs may be defined using the operation symbols of **ZFR**, the formula *Funct* is intended to abbreviate the usual formula of **ZF** which does not mention **Ap**.

The author has discussed the differences between functions and operations before; see, e.g., [4, pp. 40–42]. There the case is given for making a careful distinction. Nevertheless, one may consider the axiom asserting that every function is an operation:

(FO) $\mathit{Funct}(f) \leftarrow \forall x \forall y (\langle x, y \rangle \in f \leftrightarrow \mathbf{Ap}(f, x) = y)$.

Here we have written $\mathbf{Ap}(f, x) = y$ instead of $f(x) = y$ to avoid any possible confusion about what is meant. **FO** (which stands for "functions are operations") is a rather strong statement in that it requires that the domain of the operation be the same as the domain of the function. Under Church's thesis the domains of operations should be r.e., while domains of functions could be arbitrary sets. If we use classical logic, then **FO** conflicts with Church's thesis for a more fundamental reason, namely that there will be lots of nonrecursive set-theoretic functions. To put the matter in the language of computer science: **FO** may be appropriate for denotational semantics, but not for operational semantics.

One may say that acceptance of **FO** is a major point distinguishing classical mathematics from constructive. It is not the only one, however, since **FO** does not imply the law of the excluded middle. One might say instead that it is the principal point distinguishing the classical viewpoint from the computational viewpoint. A function, in the classical sense, need not be computable ; but an operation must be computable.

That viewpoint leads naturally enough to the desire to be able to form the set $\mathbf{op}(a, b)$ of all operations from a to b. Indeed, the designers of programming languages want to have a type $A \rightarrow B$, and both Feferman and Martin-Löf have incorporated such a construction in their theories. One cannot, however, prove the existence of $\mathbf{op}(a, b)$ in **ZFR**. Indeed, $\mathbf{op}(A, B)$ may contain arbitrarily complicated procedures. For example, given any set α, one can construct a procedure leading from A to B which, although it incorporates α (in its "code", so to speak), simply ignores α when it runs, and produces a constant value. Such procedures will have arbitrary rank, and so, in a theory in which sets must be well-founded, one cannot collect them all into a set.

Reflection will convince one that the existence of $\mathbf{op}(a, b)$ is too much to ask; we do not even know if it is consistent with **ZFR**. $\mathbf{op}(a, b)$ is surely not what the

computer scientists mean when they talk about the type $a \to b$. What one wants of $A \to B$ is not that it contains all operations from A to B, but that there be an operation $\lambda(a)$ which "abstracts" from the definition of an operation f and a set a to another operation $\lambda(a,f)$, such that $\lambda(a,f)x = fx$ for $x \in A$. Usually we write $\lambda(a,f)x$ as $\lambda x \in a.fx$. We now formulate a sensible axiom of exponentiation:

(Exp) $S(a \to b) \wedge (\forall x \in a(fx \in b) \to (\lambda(a,f) \in (a \to b) \wedge \forall x \in a(\lambda(a,f)x = fx))$

$\wedge f \in (a \to b) \to \forall x \in a(fx \in b)$.

This version of exponentiation is consistent with **ZFR**. In fact, it is implied by the principle **FO** discussed above. A natural candidate for λ is the "graph" operation Gr which associates to every set a and operation f the graph of f on a, namely $\{\langle x, fx\rangle : x \in a\}$. The operation Gr is easily defined in terms of **im**;

$$Gr(a,f) = \mathbf{im}(a, \lambda x.\langle x, fx\rangle).$$

If one assumes **FO**, then the set of graphs of operations from a to b coincides with the set of single-valued subsets of $a \times b$, which can be formed by powerset and separation.

Strictly speaking, we cannot say that **FO** implies the axiom of exponentiation since that axiom involves a new symbol λ. What we can prove can be precisely stated as follows.

8.1. Lemma. **IZFR+FO+Exp** *can be interpreted in* **IZFR+FO** *by interpreting* λ *as* Gr. *In particular, every model of* **IZFR+FO** *can be expanded to a model of* **Exp** *as well.*

Proof. We have to tell how to interpret the two operations involved in **Exp**, namely λ and the operation $\to$ involved in forming $a \to b$. The interpretation of λ is Gr, and the interpretation of $\to$ is given by

$$a \to b = \{y \in \mathcal{P}(a \times b) : \forall x, y, z(\langle x, y\rangle \in u \wedge \langle x, z\rangle \in u \to y = z)\}.$$

FO is used in verifying the second part of **Exp** under this interpretation, namely that every member of $a \to b$ is an operation from a to b. □

9. Consistency of ZFR

In this section we prove that **ZFR** is consistent. In essence we construct a model of **ZFR**. The idea for the construction goes back to Feferman [15],[3] where models are constructed for the combinatory axioms whose universe is a model to set theory M, and in which some prespecified functions $f: M \to M$ are representable. The idea is to assign some members of M to serve as indices for the prespecified functions,

[3] Feferman's construction, in turn, can be traced back through [37] to its ultimate roots in Kleene [30], where an inductive definition of an appplication relation is a key point of the theory.

and define the application relation *App* on M inductively, making $\mathbf{k}x$ be some trivial function such as $\langle 1, x\rangle$ and $\mathbf{k}xy$ be y at the basis stage, along with making the codes of the prespecified functions behave as desired. We also make $\mathbf{s}x$ and $\mathbf{s}xy$ equal to some trivial values. Then at the inductive stages, suppose we have already put $xy = u$ and $xz = v$ and $uv = w$. Then we put $\mathbf{s}xyz = w$. This inductive definition will close off after ω stages and produce a model of the combinatory axioms.

This method applies immediately to produce a model of some of the axioms of **ZFR** since we can take union, pairing, powerset as prespecified functions, as well as the operations $\mathbf{c}_\phi$ connected with separation (since the application relation is not mentioned in ϕ). However, the operation **im** cannot be prespecified since it depends on the application relation. Evidently, we cannot define $\mathbf{im}(a, f)$ until we have defined $f(x)$ for all $x \in a$. This means that our inductive definition may not close off at any stage represented by an ordinal of M. On the other hand, the application relation we are defining has to be definable in M if the axioms of set theory (in particular induction) are to be valid.

Nevertheless, if one pushes ahead with the plan of extending Feferman's construction to the transfinite, it turns out to work. What follows are just the details of this construction. Rather than present it model-theoretically, we prefer to define an interpretation from **IZFR** into **IZF**, in order to determine an upper bound for the proof-theoretic strength of **IZFR** as well as prove its consistency. Of course, it follows from such an interpretation that **ZFR** can be interpreted in classical **ZF**, and hence is consistent.

It turns out to be just as easy to make the model satisfy the axiom **FO** discussed in the previous section; this shows that our model is far from an "operational semantics" of **IZFR**, in that (if interpreted classically) it will include nonrecursive operations. By Lemma 8.1, this will automatically result in a model which also satisfies the axiom of exponentiation.

Before beginning the proof, we need some technical preparations. First, we describe a theory **IZFR***, which is a version of **IZFR** formulated with a 3-ary predicate **App** instead of using the logic of partial terms. The passage from **IZFR** to **IZFR*** is a special case of the general method of Beeson [3] for interpreting any theory in **LPT** into the first-order predicate calculus; the method associates to every function symbol f of a theory in **LPT** a relation symbol for the graph of the function. Here we take f to be **Ap**, the graph of which is **App**. We then have the axiom

$$\mathbf{App}(f, x, y) \wedge \mathbf{App}(f, x, z) \to y = z.$$

Each of the axioms of **IZFR*** is the natural translation of the corresponding axiom of **IZFR**. For instance, for pairing we have

$$\exists z, u(\mathbf{App}(\mathbf{p}, x, z) \wedge \mathbf{App}(z, y, u) \wedge \forall q(q \in u \leftrightarrow q = x \wedge q = y)).$$

For details of the reduction of **IZFR** to **IZFR***, see [3]. It follows from the proof there that **IZFR** and **IZFR*** have the same theorems in the language of **IZF**.

The second technical preparation we need is satisfaction relations for formulae of bounded complexity. Let the "complexity" of a formula be defined so that the complexity of atomic formulae is 0 and each logical operator or quantifier increases the complexity by 1. Let n be a fixed integer. Then there is a definable satisfaction relation for formulae of complexity not exceeding n. That is, there is a formula *Sat* with two free variables such that if ϕ is a formula with free variables among $x_1, \ldots, x_k$, then

$$Sat(\overline{'\phi'}, \langle x_1, \ldots, x_k \rangle) \leftrightarrow \phi(x_1, \ldots, x_k)$$

is provable (in a weak set theory, say **IZF** without either collection or replacement). These satisfaction relations are constructed explicitly in [4, Chapter XIV].

We now assign to each term t and formula ϕ of **IZFR***, a corresponding term t^* or formula ϕ^* of T; * preserves the logical operations and quantifiers.

$(t \in q)^*$	is $t^* \in q^*$	$N(t)^*$	is $t^* \in \omega$
$(t = q)^*$	is $t^* = q^*$	$S(t)^*$	is $t^* = t^*$
$\mathbf{k}^*$	is $\langle 1, 0 \rangle$	$\emptyset^*$	is $\emptyset$
$\mathbf{s}^*$	is $\langle 1, 1 \rangle$	N^*	is ω
$\mathbf{p}^*$	is $\langle 1, 2 \rangle$	$\mathbf{c}_\phi^*$	is $\langle 1, 6, '\phi' \rangle$
$\mathbf{u}^*$	is $\langle 1, 3 \rangle$	$\mathbf{im}^*$	is $\langle 1, 7 \rangle$
$\mathbf{d}^*$	is $\langle 1, 4 \rangle$	P^*	is $\langle 1, 8 \rangle$
$\mathbf{s}_N^*$	is $\langle 1, 5 \rangle$	0^*	is 0

The conditions given so far suffice to determine ϕ^* if ϕ does not contain **App**. In addition we want

$$\mathbf{App}(t, q, r)^* \text{ is } App(t^*, q^*, r^*)$$

where *App* is a formula yet to be defined. We have to find a formula *App* satisfying the following inductive conditions.

9.1. Conditions for *App*

(1) $App(\mathbf{k}^*, x, \langle 1, 0, x \rangle)$, $App(\langle 1, 0, x \rangle, y, x)$;

(2) $App(\mathbf{s}^*, x, \langle 1, 1, x \rangle)$, $App(\langle 1, 1, x \rangle, y, \langle 1, 1, x, y \rangle)$;

(3) $App(x, z, u) \wedge App(y, z, v) \wedge App(u, v, w) \rightarrow App(\langle 1, 1, x, y \rangle, z, w)$;

(4) $App(\mathbf{p}^*, x, \langle 1, 2, x \rangle)$, $App(\langle 1, 2, x \rangle, y, \{x, y\})$;

(5) $App(\mathbf{u}^*, x, \bigcup(x))$;

(6) $App(\mathbf{c}_\phi^*, a, \{x \in a : \phi^*(x)\})$ if ϕ has only x free,
$App(\mathbf{c}_\phi^*, a, \langle 1, 6, '\phi', a \rangle)$ if ϕ has $x, y_1, \ldots, y_m$ free,
$App(\langle 1, 6, '\phi', a, y_1, \ldots, y_k \rangle, y_{k+1}, \langle 1, 6, '\phi', a, y_1, \ldots, y_{k+1} \rangle)$
for $k+1 < m$, including $k = 0$, where ϕ has $x, y_1, \ldots, y_m$ free,
$App(\langle 1, 6, '\phi', a, y_1, \ldots, y_{m-1} \rangle, y_m, \{x \in a : \phi^*(x, y_1, \ldots, y_m)\})$;

(7) $App(\mathbf{s}_N^*, x, x \cup \{x\})$;

(8) $App(\mathcal{P}^*, x, \mathcal{P}(x))$;

(9) $App(\mathbf{d}^*, n, \langle 1, 4, n\rangle)$, $App(\langle 1, 4, n\rangle, m, \langle 1, 4, n, m\rangle)$,
$App(\langle 1, 4, n, m\rangle, x, \langle 1, 4, n, m, x\rangle)$,
$App(\langle 1, 4, n, m, x\rangle, y, \{z : (n = m \wedge z \in x) \wedge (n \neq m \wedge z \in y)\})$;

(10) $App(\mathbf{im}^*, a, \langle 1, 7, a\rangle)$,
$\forall x \in a \exists y \in u App(f, x, y) \wedge \forall y \in u \exists x \in a App(f, x, y) \rightarrow App(\langle 1, 7, a\rangle, f, u)$;

(11) $Funct(f) \wedge \langle x, y\rangle \in f \rightarrow App(f, x, y)$;

(12) If ψ is any formula of **IZF** satisfying the above conditions with ψ in place of *App*, then $App(f, x, y) \rightarrow \psi(f, x, y)$.

Clause (11) is needed only to model **FO**; otherwise it may be omitted. Clause (12) says that *App* satisfies every instance of induction on the definition of *App* that can be expressed in **IZF**.

9.2. Theorem. **ZFR** *is consistent.*

Proof. Fix an integer n. Restricting condition (6) to formulae of complexity less than n, Conditions 9.1 can be expressed in **IZF**. By standard methods, we can find a formula *App* satisfying these conditions. It is easy to verify that every axiom of **ZFR** except the axiom of images is satisfied under the translation ϕ^* (where separation is restricted to formulae of complexity less than n). The axiom

$$\mathbf{App}(f, x, y) \wedge \mathbf{App}(f, x, z) \rightarrow y = z$$

is verified by induction on the definition of *App*, using clause (12); note that we have taken care that the f in various clauses have different forms to prevent a conflict; in particular none of the "codes" in clauses (1) to (10) is a function, so there is no conflict with clause (11). Similarly, the combinatorial axioms are satisfied because *App* is solution of the inductive conditions (1) to (3).

Now consider the axiom of images. Suppose $\forall x \in a \exists y\, App(f, x, y)$. By collection we can find a set v such that $\forall x \in a \exists y \in v\, App(f, x, y)$. By separation we can form $u = \{y \in v : \exists x \in a\, App(f, x, y)\}$. It follows from clause (10) that $App(\{1, 7, a\rangle, a, u)$ and $App(\mathbf{im}^*, a, \langle 1, 7, a\rangle)$, verifying the axiom of images. □

The reader who finds the treatment of inductive definitions in this proof somewhat sketchy will find sufficient detail in the next section, where we refine the construction to show it can be done using only replacement.

9.3. Theorem. *Every formula in the language of* **ZF** *which is provable in* **ZFR** *is provable in* **ZF**.

Proof. Let ϕ be a formula in the language of **ZF** which is a theorem of **ZFR**. Let n be an integer large enough that there is a derivation of ϕ involving only formulae of complexity less than n. By induction on the length of derivations, every derivation

involving only formulae of complexity less than n of a formula A can be transformed into a derivation in **ZF** of A^*. But in case A is in the language of **ZF**, A^* is just A. □

In the remainder of this section, we shall consider several possible extensions of **ZFR**.

9.4. Everything is an integer or a set and not both. The consistency proof above actually shows the consistency of **ZFR** with the collection schema and the axiom $\forall x S(x)$. Should we require the consistency of **ZFR** with $\forall x((N(x) \vee S(x)) \wedge \neg(N(x) \wedge S(x)))$, the proof may be supplemented by the methods of the exercises of [4, Chapter VIII].

9.5. Pairing and projection functions. One may, of course, define ordered pairs as usual from the unordered pairing function **p**, namely $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$. Define $\mathbf{p}_0 = \lambda z.\mathbf{u}(\bigcap(z))$ where $\bigcap(z) = \{x \in \mathbf{u}(z) : \forall y \in z. x \in y\}$. (This definition of intersection is all right for nonempty sets.) Then $\mathbf{p}_0(\langle x, y\rangle) = x$, so we have defined a left projection function. The right projection can be defined as $\mathbf{p}_0 = \lambda z.\{w \in \mathbf{u}(\mathbf{u}z) : \forall y \in \mathbf{u}z(z = \langle \mathbf{p}_0 z, y\rangle \to w \in y)\}$. These definitions are intuitionistically correct; some simpler definitions work classically but not intuitionistically. Hence, it was theoretically unnecessary to include constants for pairing and projection in **IZFR**.

9.6. Adding constants for specific set-theoretically definable operations. In this case the consistency proof will still apply; there are just more basis clauses in the inductive definition. For example, by 9.5 we could add constants π, $\mathbf{j}_0$, and $\mathbf{j}_0$ with the axioms $\mathbf{j}_0(\pi xy) = x$ and $\mathbf{j}_0(\pi xy) = y$.

9.7. Strong exponentiation. We might wish to assert the existence of the set $\mathbf{op}(A, B)$ of all operations from A to B. Note that this is not the same as the set of all functions from A to B, which can be defined using powerset and separation. Neither is it the same as the set $A \to B$, as discussed in Section 8. To be precise, we consider the axiom

(OPEXP) $S(\mathbf{op}(a, b)) \wedge \forall f(f \in \mathbf{op}(a, b) \leftrightarrow \forall x \in a(fx \in b))$.

Note that **OPEXP** fails in the model of **ZFR** constructed above, since even $\mathbf{op}(\mathbf{N}, \mathbf{N})$ does not exist, in view of the fact that $\lambda n.\mathbf{k}0x$ (which is actually $\mathbf{s}(\mathbf{s}(\mathbf{kk}, \mathbf{k}0), \mathbf{k}x)$) would have to belong to $\mathbf{op}(\mathbf{N}, \mathbf{N})$ for every set x; but this is impossible since $\mathbf{k}x$ may have arbitrarily large rank, as it is interpreted in the model. We do not known if **ZFR** + **OPEXP** is consistent.

This may seem like a drawback from the point of view of computer science since the type construction $A \to B$ is important. However, the version of exponentiation just considered is too strong. The right version of exponentiation, similar to the version in Martin-Löf's theories, is the axiom **Exp** discussed in Section 8; and we have already seen that this follows from **FO** and hence holds in the model constructed above.

10. IZFR is no stronger than IZF-Rep

This section has two purposes:

(1) for those readers not on intimate terms with definition by transfinite induction in set theory, it will provide complete details of our consistency proof; and

(2) for those readers interested in constructive set theory, we present a refinement of the proof which shows that we can get by with only replacement, instead of collection, which was needed in the proof presented above. (The two are not constructively equivalent, as discussed in Section 7.)

The idea of the proof is to replace the definition of *App* by an inductive definition of a four-place relation R, where intuitively, if $R(f, x, y, u)$, then u is the "reason" why $App(f, x, y)$. We will write down suitable conditions for the definition of R, and afterwards define $App(f, x, y) \leftrightarrow \exists u R(f, x, y, u)$. The reason this is useful is that the definition of R is essentially by induction on $\in$, while that of *App* is not. In formalizing the definition of *App*, we need collection to collect together the u values such that $R(f, x, y, u)$ for $x \in a$, in order to define $\mathbf{im}(f, a)$. Defining R instead permits us to get by with replacement.

10.1. Conditions for R

(1) $R(\mathbf{k}^*, x, \langle 1, 0, x\rangle, 0)$, $\quad R(\langle 1, 0, x\rangle, y, x, 0)$;

(2) $R(\mathbf{s}^*, x, \langle 1, 1, x\rangle, 0)$, $\quad R(\langle 1, 1, x\rangle, y, \langle 1, 1, x, y\rangle, 0)$;

(3) $R(\mathbf{p}^*, x, \{1, 2, x, 0)$, $\quad R(\langle 1, 2, x\rangle, y, \{x, y\}, 0)$;

(4) $R(\mathbf{u}^*, x, \bigcup(x), 0)$;

(5) $R(\mathbf{c}^*_\phi, a, \{x \in a : \phi^*(x)\}, 0)$ if ϕ has only x free,

$R(\mathbf{c}^*_\phi, a, \{1, 6, '\phi', a\rangle, 0)$ if ϕ has $x, y_1, \ldots, y_m$ free,

$R(\langle 1, 6, '\phi', a, y_1, \ldots, y_k\rangle, y_{k+1}, \langle 1, 6, '\phi', a, y_1, \ldots, y_{k+1}\rangle, 0)$ for $k+1 < m$, including $k = 0$, where ϕ has $x, y_1, \ldots, y_m$ free,

$R(\langle 1, 6, '\phi', a, y_1, \ldots, y_{m-1}\rangle, y_m, \{x \in a : \phi(x, y_1, \ldots, y_m)\}, 0)$;

(6) $R(\mathbf{s}^*_{\mathbf{N}}, x, x \cup \{x\}, 0)$;

(7) $R(\mathcal{P}^*, x, \mathcal{P}(x), 0)$;

(8) $R(\mathbf{d}^*, n, \langle 1, 4, n\rangle, 0)$, $\quad R(\langle 1, 4, n\rangle, m, \langle 1, 4, n, m\rangle, 0)$,

$R(\langle 1, 4, n, m\rangle, x, \langle 1, 4, n, m, x\rangle, 0)$,

$R(\langle 1, 4, n, m, x\rangle, y, \{z : (n = m \wedge z \in x) \vee (n \neq m \wedge z \in y)\}, 0)$;

(9) $R(x, z, u, a) \wedge R(y, z, v, b) \wedge R(u, v, w, c) \rightarrow R(\langle 1, 1, x, y\rangle, z, w, \langle u, v, a, b, c\rangle)$;

(10) $Funct(g) \wedge Funct(h) \wedge \forall x \in a \exists u, v(\langle x, u\rangle \in g \wedge \langle x, v\rangle \in h \wedge R(f, x, u, v))$
$\wedge a = Dom(g) \wedge a = Dom(h) \rightarrow R(\langle 1, 7, a\rangle, f, Ran(g), \langle g, h\rangle)$;

(11) $R(\mathbf{im}^*, a, \langle 1, 7, a\rangle, 0)$;

(12) $Funct(f) \wedge \langle x, y\rangle \in f \rightarrow R(f, x, y, 0)$;

(13) If ψ is any formula of IZF satisfying the above conditions with ψ in place of R, then $R(f, x, y, u) \rightarrow \psi(f, x, y, u)$.

The last condition says that R satisfies every instance of induction on the definition of R that can be expressed in IZF. (This is not the same as saying it is the least

solution of the inductive condition.) Note that if we restrict the complexity of ϕ in (5) to be less than a fixed integer n, then conditions (1)–(12) can be expressed by a single formula of **IZF**, with an addition predicate symbol R, by means of the satisfaction-definitions discussed above. We shall refer to these conditions, plus (13) for all ψ, as "10.1_n".

10.2. Lemma. *Let n be a fixed integer. Suppose* **IZF-Rep** *proves Conditions* 10.1_n *for some formula R. Define* $App(f, x, y) \leftrightarrow \exists u R(f, x, y, u)$. *Then* **IZF-Rep** *proves Conditions* 9.1_n.

Proof. We argue in **IZF-Rep**. We shall show

$$R(f, x, y, u) \wedge R(f, x, w, v) \rightarrow u = v \wedge y = w. \qquad (*)$$

We shall do so by induction on the definition of R. To be precise, we will apply clause (13) of 10.1, with ψ taken to be

$$\psi(f, x, y, u) \leftrightarrow \forall w, v(R(f, x, w, v) \rightarrow u = v \wedge y = w).$$

The argument proceeds by cases.

Case 1: $R(f, x, y, u)$ because of (1)–(8) or (11) or (12). Then $u = 0$. Suppose $R(f, x, w, v)$. Suppose, for example, that $f = \mathbf{k}^*$. Then $w = \langle 1, 0, x\rangle = y$ and $u = v = 0$ (as follows from clause (1)). Similarly for the eighteen other possible forms of f.

Case 2: $R(f, x, y, u)$ because of (9). Let $R(f, x, w, v)$. Then we apply the lemma, proved using clause (13), that $R(\langle 1, 1, r, s\rangle, x, y, u)$ implies $u = \langle u', v', a, b, c\rangle$, where $R(r, x, u', a) \wedge R(s, x, v', b) \wedge R(u', v', y, c)$. Let $f = \langle 1, 1, r, s\rangle$. By the induction hypothesis, we have

$$\psi(r, x, u', a) \wedge \psi(s, x, v', b) \wedge \psi(u', v', y, c).$$

Hence, u', a, v', and b are uniquely determined by f and x. Hence, also y and c are uniquely determined by f and x. It follows that y and u are uniquely determined by f and x. Hence, $w = y$ and $u = v$ as required.

Case 3: $R(\langle 1, 7, a\rangle, f, u, \langle g, h\rangle)$ because of (10). Suppose also $R(\langle 1, 7, a\rangle, f, u', \langle g', h'\rangle)$. By (13) it can be shown that u', g', h' are as in (10) as well; that is, both g and g' are functions with domain a, as are h and h', and for all $x \in a$ we have $R(f, x, g(x), h(x))$ and $R(f, x, g'(x), h'(x))$. By induction hypothesis we then have $g(x) = g'(x)$ and $h(x) = h'(x)$ for all $x \in a$. Hence, by extensionality, $g = g'$ and $h = h'$. That completes the proof of (*).

Now let *App* be defined as in the statement of the lemma. We shall prove that *App* satisfies Conditions 9.1_n. All the conditions of 9.1 except (3) and (10) may be trivially verified by taking $u = 0$. Consider (3). Suppose $App(x, z, u) \wedge App(y, z, v) \wedge App(u, v, w)$. Then there exist a, b, c such that

$$R(x, z, u, a) \wedge R(y, z, v, b) \wedge R(u, v, w, c).$$

Hence, by 10.1(9), we have $R(\langle 1, 1, x, y\rangle, z, w, \langle u, v, a, b, c\rangle)$. Hence, $\exists u R(\langle 1, 1, x, y\rangle, z, w, u)$. Hence, $App(\langle 1, 1, x, y\rangle, z, w)$ as required by 9.1(3).

Now consider (10). Suppose

$$\forall x \in a \exists y \in u\, App(f, x, y) \wedge \forall y \in u \exists x \in a\, App(f, x, y).$$

By (*) we have $\forall x \in a \exists! \langle y, v\rangle R(f, x, y, v)$. Applying replacement, there is a function Y and a function V, both with domain a, such that $\forall x \in a R(f, x, Y(x), V(x))$. Then by (10) of 10.2, we have $R(\langle 1, 7, a\rangle, f, Ran(Y), \langle Y, V\rangle)$. Hence, $App(\langle 1, 7, a\rangle, f, Ran(Y))$. But, by (*), we have $Y(x) \in u$ for all $x \in a$. By extensionality, $u = Ran(Y)$. Hence, $App(\langle 1, 7, a\rangle, f, u)$ as required in 9.1(10). □

10.3. Lemma. IZF-Rep *proves, for each* $a_1, \ldots, a_n$, *that there is a least transitive set* $TC(a_1, \ldots, a_n)$ *containing* $a_1, \ldots, a_n$.

Proof. Let a abbreviate $a_1, \ldots, a_n$; let $a \in w$ abbreviate $a_1 \in w \wedge \ldots \wedge a_n \in w$; let $\forall a$ abbreviate $\forall a_1 \ldots \forall a_n$. We shall use $\in$-induction to prove

$$\forall a \exists! w(Trans(w) \wedge a \in w \wedge \forall v(Trans(v) \wedge a \in v \rightarrow w \subset v))$$

where $Trans(w)$ is $\forall u, v(u \in v \wedge v \in w \rightarrow u \in w)$. Suppose

$$\forall x \in a \exists! w(Trans(w) \wedge x \in w \wedge \forall v(Trans(v) \wedge x \in v \rightarrow w \subset v)).$$

By replacement, there is a set b such that

$$\forall x \in a \exists w \in b(Trans(w) \wedge x \in w \wedge \forall v(Trans(v) \wedge x \in v \rightarrow w \subset v)).$$

Let

$$u = \{a\} \cup \{w \in b : \exists x \in a(Trans(w) \wedge x \in w \wedge \forall v(Trans(v) \wedge x \in v \rightarrow w \subset v))\}.$$

Then $Trans(u) \wedge \forall x \in a(x \in u) \wedge a \in u$. Now $TC(a)$ can be defined as the intersection of all transitive subsets of u that contain a. □

Remark. The proof of the lemma uses powerset, replacement, and $\in$-induction.

10.4. Proposition. *Let* n *be fixed. There is a formula* R *of* **IZF** *such that* **IZF-Rep** *proves Conditions* 10.1_n.

Proof. Conditions 10.1_n, (1) through (12) can be written in the form

$$\Psi(f, x, y, u, \mathbf{R}) \rightarrow \mathbf{R}(f, x, y, u)$$

for a suitable formula Ψ in the language of **IZF** with one additional four-place predicate symbol **R**. If A is a transitive set, we call the set w "*A-closed*" if $w \subset A^4$ and

$$f \in A \wedge x \in a \wedge y \in a \wedge u \in A \wedge \Psi(f, x, y, u, w) \rightarrow \langle f, x, y, u\rangle \in w.$$

We define $R(f, x, y, u)$ to be a formula expressing "$\langle f, x, y, u\rangle$ belongs to every $TC(f, x, y, u)$-closed set."

We have to prove that R satisfies the conditions 10.1_n. A key observation is the following, which is an immediate consequence of the definitions involved:

> If A and B are transitive and $B \subset A$ and w is A-closed, then $w \cap B$ is B-closed. (**)

Conditions 10.1(1) through (8), (11), and (12), can all be treated simultaneously since they all have the form $R(t, q, s, 0)$ for certain t, q, and s. Let W be A-closed, where $A = TC(t, q, s, 0)$. Then, by the definition of A-closed, we have $\langle t, q, s, 0\rangle \in W$. Hence, $R(t, q, 0)$. Now consider (9). Suppose

$$R(x, z, u, a) \wedge R(y, z, v, b) \wedge R(u, v, w, c).$$

Let $A = TC(\langle 1, 1, x, y\rangle, z, w, \langle u, v, a, b, c\rangle)$. Let W be A-closed. We have to show $\langle\langle 1, 1, x, y\rangle, z, w, \langle u, v, a, b, c\rangle\rangle \in W$. Note that x, y, z, w, u, v, a, b, and c are all members of A. Since $R(x, z, u, a)$, we have: $\langle x, z, u, a\rangle$ belongs to every $TC(x, z, u, a)$-closed set. By (**), $W \cap TC(x, z, u, a)$ is $TC(x, z, u, a)$-closed. Hence, $\langle x, z, u, a\rangle \in W$. Similarly, since $R(y, z, v, b)$, we have $\langle y, z, v, b\rangle \in w$, and since $R(u, v, w, c)$, we have $\langle u, v, w, c\rangle \in W$. Since W is A-closed, it follows that $\langle\langle 1, 1, x, y\rangle, z, w, \langle a, b, c\rangle\rangle \in w$ as required.

Now consider (10). Suppose $\forall x \in a R(f, x, g(x), h(x))$ where g and h are set-theoretic functions. (Since we are working in pure set theory, not in **ZFR**, there is theoretically no chance of confusing the abbreviation $g(x)$ with the application in **ZFR**.) Suppose $Dom(g) = Dom(h) = a$. We have to show $R(\langle 1, 7, a\rangle, f, u, \langle g, h\rangle)$. Let $A = TC(\langle 1, 7, a\rangle, f, u, \langle g, h\rangle)$, and let W be A-closed. We have to show $\langle\langle 1, 7, a\rangle, f, u, \langle g, h\rangle\rangle \in w$. It will suffice to show $\forall x \in a(\langle f, x, g(x), h(x)\rangle \in w)$. Let $x \in A$. Then $R(f, x, g(x), h(x))$; let $B = TC(f, x, g(x), h(x))$. Then $\langle f, x, g(x), h(x)\rangle$ belongs to every B-closed set, and by (**), $W \cap B$ is B-closed; hence, $\langle f, x, g(x), h(x)\rangle \in w$.

Now consider (13). Suppose the formula ψ satisfies (1)–(12) with ψ in place of R. Let $R(f, x, y, u)$. We have to show $\psi(f, x, y, u)$. Let $A = TC(f, x, y, u)$. Let $W = \{\langle a, b, c, d\rangle \in A^4 : R(a, b, c, d)\}$. Then W is A-closed and is a subset of every A-closed set, by definition of R. Define $W' = \{\langle a, b, c, d\rangle \in A : \psi(a, b, c, d)\}$. Then, by hypothesis, W' is A-closed. Hence, $W \subset W'$. Since $R(f, x, y, u)$, we have $\langle f, x, y, u\rangle \in W$. Hence, $\langle f, x, y, u\rangle \in W'$. Hence, $\psi(f, x, y, u)$. □

Remark. Separation is used in the last paragraph to form W'. If we only wanted to verify R-induction for bounded formulae, we would need only bounded separation since R can be defined by a bounded formula.

10.5. Theorem. *Every theorem of* **IZFR+FO** *in the language of* **IZF** *is provable in* **IZF-Rep**.

Proof. Let R be as in Proposition 10.4. Define $App(f, x, y) \leftrightarrow \exists u R(f, x, y, u)$. By Lemma 10.2, **IZF-Rep** proves Conditions 9.1_n. Now the translation ϕ^* given at the

beginning of Section 9 is completely defined. (Technically, one may view the translation as going from **ZFR** to an extension of **IZF-Rep** by suitable terms.) Conditions 9.1 make it clear that ϕ^* is a theorem of **IZF-Rep** for each axiom of **ZFR+FO** of complexity less than n. By induction on the length of proofs, one shows that if p is a proof in **ZFR+FO** of ϕ, each of whose lines is of complexity less than n, then ϕ^* is a theorem of **IZF-Rep**. Since n was arbitrary, we may choose it to exceed the maximum complexity of lines of a specific proof p in **ZFR** of a formula ϕ in the language of **IZF**. Since ϕ is in the language of **IZF**, ϕ^* is just ϕ. Hence, ϕ is a theorem of **IZF-Rep**. □

Remark. The theorem applies just as well to **ZFR** plus replacement as to **ZFR**. We conjecture that **ZFR** does not prove the replacement schema. The theorem leaves open the question of the exact proof-theoretic strength of **ZFR**.

11. Operational semantics of ZFR

The model constructed above is the natural set-theoretic model of **ZFR**; it gives the denotational semantics. But it includes many nonrecursive operations from N to N (at least if classical logic is used), and hence does not provide an operational semantics. Of course, using classical logic, there are bound to be nonrecursive *functions*, but as it turns out, it is only possible to define recursive *operations*. We shall construct a *natural recursion-theoretic model* of **ZFR**, in which all operations from N to N are recursive.

The model is constructed by the same technique as above, that is, by inductively defining an application relation. We shall have to make two modifications in the construction. The first one is obvious: we shall leave out clause (11) in 9.1, which makes every set-theoretic function an operation. With classical logic, that clause will lead to many nonrecursive functions. We want to construct instead the model which has only those operations that must be there on account of the axioms.

The second modification we have to make is not quite so obvious. It has to do with the representation of the integers as the "Von Neumann integers" of set theory. **ZFR** takes integers as primitive, and does not use the Von Neumann representation. Our first model of **ZFR** did model the integers as Von Neumann integers, but we shall have to abandon the Von Neumann integers to get a model with only recursive operations. Here is why.

When we ask our computation system to produce an integer, we want to get a numeral as answer, not a definition of some set like $q = \{x : x = \emptyset \wedge P\}$, where P is an unsolved problem like Fermat's last theorem. One may believe—in fact, if one takes classical logic, one must believe—that this definition defines either zero or one, even though we do not know which. But such a definition is not the kind of answer we want from the computer as an integer. If we accept the Von Neumann integers as the definition of "integer", we will be stuck with "integers" like this one.

It is for this reason that **ZFR** takes the integers as primitive, instead of as defined in set theory. If we accept the Von Neumann integers as the integers, then we will certainly have nonrecursive operations on the integers, for example,

$$f(n) = \{m : m = 0 \wedge \{n\}(n)\downarrow\}$$

which defines the characteristic function of the halting problem if we use classical logic. (With only intuitionistic logic, we cannot prove that the values of f are Von Neumann integers; to do that we have to prove that $f(n)$ is either zero or one.) Note that the definition of f does not work if integers are primitive; it depends on the Von Neumann construction of integers as sets.

One will naturally wonder at this point, "but are the Von Neumann integers not isomorphic to the integers?" The answer is that they are isomorphic by a set-theoretic function, but not necessarily by an operation. If we had an operation g that computed the integer equivalent to a given Von Neumann integer, we could solve Fermat's last theorem by applying g to the set q constructed above. Of course, this remark will only become a precise theorem about the nonexistence of an isomorphism operation *after* we have constructed a model with only recursive operations. But it illustrates the situation. One can use the recursion theorem to embed the integers in the Von Neumann integers:

$$f(n) = \mathbf{d}(0, h, \emptyset, f(\mathbf{p}_N n) \cup \{f(\mathbf{p}_N n)\}),$$

but one cannot construct the inverse of f and show it to be defined on all the Von Neumann integers.

We are suggesting a viewpoint according to which, even with classical logic, the set q defined above does not count as an integer. This viewpoint is codified in **ZFR**. It is at odds with the set-theoretic tradition, but it is the viewpoint suitable for computational mathematics, whether or not one accepts classical logic. The main theorem of this section, that there is a natural recursion-theoretic model of **ZFR** in which only recursive operations from N to N occur, shows that this viewpoint is coherent.

With these explanations finished, we can proceed to the construction of the model. This construction can best be understood if we think of it in two stages. First we have to define a model of the set-theoretic part of **ZFR** in which there will be three kinds of things: sets, numbers, and other objects, which are clearly labelled. Having done that, we shall then give an inductive definition of the application relation, much as we did above.

The model, which we shall call M, will be built of sets since set theory is the traditional metatheory of mathematics. It will contain only sets of the three forms $\langle 0, x\rangle$, $\langle 2, x\rangle$, and $\langle 1, x\rangle$. That is, only ordered pairs whose first component is 0, 2, or 1; this number will tell us whether the object in question is a number, set, or other object. The exact definition of the universe of M is as follows.

11.1. Definition. The universe, sets, and integers of M are defined inductively by

the following clauses:

(1) The elements of M are the sets in M, the numbers in M, and the objects in M.

(2) The numbers of M are all sets of the form $\langle 0, n\rangle$ with $n \in \omega$.

(3) If x is a set of elements of M, then $\langle 2, x\rangle$ is a set in M.

(4) $\mathbf{k}^*$, $\mathbf{s}^*$, $\mathbf{p}^*$, $\mathbf{c}_\phi^*$, $\mathbf{u}^*$, $\mathbf{im}^*$, $\mathbf{d}^*$, $\mathbf{s}_\mathbf{N}^*$, $\mathcal{P}^*$, as defined in Section 9, are objects in M.

(5) If n and m are numbers in M, a is a set in M, and x and y are any elements of M, then the following are objects in M; for intelligibility we also give their intended denotations:

$\langle 1, 0, x\rangle$ to denote $\mathbf{k}x$,
$\langle 1, 1, x\rangle$ to denote $\mathbf{s}x$,
$\langle 1, 1, x, y\rangle$ to denote $\mathbf{s}xy$,
$\langle 1, 2, x\rangle$ to denote $\mathbf{p}x$,
$\langle 1, 4, n\rangle$ to denote $\mathbf{d}n$,
$\langle 1, 4, n, m\rangle$ to denote $\mathbf{d}nm$,
$\langle 1, 4, n, m, x\rangle$ to denote $\mathbf{d}mnx$.

(6) If ϕ has $x, y_1, \ldots, y_m$ free with $m > 0$ and $k+1 < m$, then the following are objects in M:

$\langle 1, 6, '\phi', a\rangle$ to denote $\mathbf{c}_\phi a$,
$\langle 1, 6, '\phi', a, y_1, \ldots, y_k\rangle$ to denote $\mathbf{c}_\phi a y_1 \ldots y_k$.

As in Section 9, we can cast the construction of M conveniently as a translation from **IZFR*** into a suitable extension by terms of **IZF**, in which the fact that M satisfies ϕ will be expressed by a formula ϕ^*. We define $\emptyset^* = \langle 2, \mathbf{N}^* = \langle 2, \{\langle 0, n\rangle : n \in \omega\}\rangle$, $S(t)^*$ is $S^*(t^*)$, where S^* is a formula defining the sets in M. We define $(t = q)^*$ to be $t^* = q^*$.

11.2. Definition. Membership in M: $(x \in a)^*$ is the formula

$$S^*(a) \wedge \exists b(a = \langle 2, b\rangle \wedge x \in b).$$

It will also be convenient to write $x \in_M a$ for $(x \in a)^*$.

The above definitions suffice to determine ϕ^* for all formulae ϕ not containing **App**. In addition we shall define, as in Section 9,

$$\mathbf{App}(t, q, r)^* \text{ is } App(t^*, q^*, r^*)$$

where *App* is a formula yet to be defined. We have to find a formula *App* satisfying the following inductive conditions.

11.3. Definition. The application relation of M is defined inductively by the following clauses:

(1) $App(\mathbf{k}^*, x, \langle 1, 0, x\rangle)$, $App(\langle 1, 0, x\rangle, y, x)$;

(2) $App(\mathbf{s}^*, x, \langle 1, 1, x\rangle)$, $App(\langle 1, 1, x\rangle, y, \langle 1, 1, x, y\rangle)$;

(3) $App(x, z, u) \wedge App(y, z, v) \wedge App(u, v, w) \rightarrow App(\langle 1, 1, x, y\rangle, z, w)$;

(4) $App(\mathbf{p}^*, x, \langle 1, 2, x\rangle)$, $App(\langle 1, 2, x\rangle, y, \langle 2, \{x, y\}\rangle)$;

(5) $App(\mathbf{u}^*, \langle 2, a\rangle, \langle 2\rangle \cup \{x : \langle 2, x\rangle \in a\}))$;

(6) $App(\mathbf{c}^*_\phi, \langle 2, a\rangle, \langle 2, \{x \in a : \phi^*(x)\}\rangle)$ if ϕ has only x free,
$App(\mathbf{c}^*_\phi, \langle 2, a\rangle, \langle 1, 6, '\phi', a\rangle)$ if ϕ has $x, y_1, \ldots, y_m$ free,
$App(\langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_k\rangle, y_{k+1}, \langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_{k+1}\rangle)$
for $k+1 < m$, including $k = 0$, where ϕ has $x, y_1, \ldots, y_m$ free,
$App(\langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_{m-1}\rangle, y_m, \langle 2, \{x \in a : \phi^*(x, y_1, \ldots, y_m)\}\rangle)$;

(7) $App(\mathbf{s}^*_{\mathbf{N}}, \langle 0, n\rangle, \langle 0, n \cup \{n\}\rangle)$;

(8) $App(\mathscr{P}^*, \langle 2, x\rangle, \langle 2, \{\langle 2, z\rangle : z \in \mathscr{P}(x)\}\rangle)$;

(9) $N(n)^* \to App(\mathbf{d}^*, n, \langle 1, 4, n\rangle)$,
$N(n)^* \wedge N(m)^* \to App(\langle 1, 4, n\rangle, m, \langle 1, 4, n, m\rangle)$,
$N(n)^* \wedge N(m)^* \to App(\langle 1, 4, n, m\rangle, x, \langle 1, 4, n, m, x\rangle)$,
$N(n)^* \to App(\langle 1, 4, n, n, x\rangle, y, x)$,
$N(n)^* \wedge N(m)^* \wedge n! = m \to App(\langle 1, 4, n, m, x\rangle, y, y)$;

(10) $S(b)^* \to^* App(\mathbf{im}^*, b, \langle 1, 7, b\rangle)$,
$S^*(b) \wedge \forall x \in_M b \exists y \in_M u App(f, x, y) \wedge \forall y \in_M u \exists x \in_M b App(f, x, y)$
$\to App\langle 1, 7, b\rangle, f, u)$

(11) If ψ is any formula of **IZF** satisfying the above conditions with ψ in place of *App*, then $App(f, x, y) \to (f, x, y)$.

11.4. Theorem. *A formula App meeting the above conditions can be constructed; the resulting translation of ϕ to ϕ^* is sound. Hence a model M of* **ZFR** *exists in which the application relation satisfies the above inductive conditions.*

Proof. The inductive definition can be formalized as in Section 9. We then check that the axioms of **ZFR** are satisfied; this goes as in Section 9 except that the definition of membership in M is slightly different. Let us check, for example, the pairing axiom. Suppose M satisfies $(z \in \mathbf{p}xy)$. That is,

$$\exists u \exists v (App(\mathbf{p}, x, u) \wedge App(u, y, v) \wedge z \in v).$$

Then $u = \langle 1, 2, x\rangle$, and $v = \langle 2, \{x, y\}\rangle$. But $(z \in \langle 2, \{x, y\}\rangle)^*$ is $z \in \{x, y\}$. Hence, M satisfies $(z \in \mathbf{p}xy)$ iff $z = x$ or $z = y$, which is the same as $(z = x \vee z = y)^*$.

The axiom $App(x, y, u) \wedge App(x, y, v) \to u = v$ holds in M, as one verifies by induction on the definition of *App*, exactly as in Theorem 9.2.

We now check the axioms of images. Suppose b is a set in M, and suppose f is an element of M such that fx is defined for every $x \in_M b$. Let $u = \langle 2, \{v : \exists x \in_M b(App(x, y))\}\rangle$. Then $\forall x \in_M b \exists y \in_M u App(x, y)$ and also $\forall y \in_M u \exists x \in_M b App(x, y)$. Hence, by the definition of *App*, we have $App(\langle 1, 7, b\rangle, f, u)$. Note that the set u can be formed using replacement since *App* is single-valued.

We now check the axiom of separation. For simplicity we consider only the case in which ϕ contains only x free. Suppose $\langle 2, a\rangle$ is a set in M. Then in M, $\mathbf{c}_\phi(\langle 2, a\rangle)$ is interpreted as $\langle 2, \{x \in a : \phi^*(x)\}\rangle$, where $\phi^*(x)$ is the formula expressing that M satisfies ϕ. Hence M satisfies $x \in \mathbf{c}_\phi(a)$ if and only if M satisfies $x \in a \wedge \phi(x)$.

We now check the axiom of union. Suppose $\langle 2, a\rangle$ is a set in M. Then $\mathbf{u}\langle 2, a\rangle$ is interpreted in M as 2, $\bigcup\{x: \langle 2, x\rangle \in a\}\rangle$. Thus M satisfies $z \in \mathbf{u}\langle 2, a\rangle$ iff $z \in \bigcup\{x: \langle 2, x\rangle \in a\}$ iff $\exists x(\langle 2, x\rangle \in a \wedge z \in x)$ iff $\exists x(\langle 2, x\rangle \in a \wedge z \in_M \langle 2, x\rangle)$. Since every set in M has the form $\langle 2, x\rangle$, this is equivalent to $\exists w(w \in a \wedge z \in_M x)$. This in turn can be written $\exists w(w \in_M \langle 2, a\rangle \wedge z \in_M x)$, which is the condition mentioned in the union axiom.

The other axioms can be checked similarly. □

At this point it is instructive to see intuitively why the characteristic function of the halting problem cannot be defined in M. Let us try defining $f(n) = \langle 0, \{m: m = 0 \wedge \{n\}(n)\downarrow\}$. That would be the object playing the role of characteristic function of the halting problem in M. But there is no operation f of M which has this behavior under the *App* relation. (That has not been proved yet, of course, but just try to construct one.)

Our next goal is to actually prove that the application relation of M has the property that it introduces only recursive operations on the integers. One obviously has to prove something by induction on the definition of the application relation, and that something has to refer to arbitrary operations in the model, and not only to operations from N to N.

The key tool in the proof is the notion of term reduction and the related notion of term models. The basic facts can be found in [4, Chapter V]. There one finds the reduction rules for the combinatory part of **ZFR**, and the definition of "inside-first reduction", which is better-known in computer science as "call-by-value" reduction; it just means that one must evaluate all subterms of the term one is evaluating, even if they are only to be thrown away.

We shall construct a combinatory algebra A from M, intuitively by identifying all the sets in M to a single object, and making the other identifications that this necessitates. It turns out to be easier to describe A directly, and then to give a homomorphism of combinatory algebras from M to A. A is constructed as follows: It is the normal-term model whose elements are closed normal terms in the language of **ZFR** augmented by a new constant **a**. The application relation in this model is defined by $Ap(t, s) = RED(ts)$, where $RED(t)$ is the result of reducing the term t by call-by-value term reduction, using the following reduction rules (here written as equations):

$$\mathbf{k}xy = x, \qquad \mathbf{s}xyz = xz(yz),$$

$$\mathbf{d}\overline{nn}xy = x, \qquad \mathbf{d}\overline{n}\overline{m}xy = y \text{ if } n \neq m,$$

$$\emptyset = \mathbf{a}, \qquad \mathbf{p}xy = \mathbf{a},$$

$$\mathbf{u}\mathbf{a} = \mathbf{a}, \qquad \mathcal{P}\mathbf{a} = \mathbf{a}, \qquad \mathbf{im}(\mathbf{a}, x) = \mathbf{a},$$

$$\mathbf{c}_\phi(\mathbf{a}, y_1, \ldots, y_m) = \mathbf{a} \quad \text{if } \phi \text{ contains exactly } x, y_1, \ldots, y_m \text{ free.}$$

The exact definition of *RED* can be given as in [4, p. 113], but using the above reduction rules.

11.5. Lemma. *A is a combinatory algebra. If we interpret $N(x)$ as the numerals, then A also satisfies the axioms for* **d**, *and all functions from N to N representable in A are recursive.*

Proof. The construction of normal-term models is carried out in detail in [4, Chapter VI]; nothing new is involved here. Since call-by-value reduction is recursive, if t represents a function from N to N, then the value of t at n can be computed just by reducing $t\bar{n}$ by call-by-value reduction. □

We now define a homomorphism of combinatory algebras from M to A. The definition is by induction on the definition of elements of M.

11.6. Definition. To each element x of M we associate an element x° of A as follows:
(1) if x is a set in M, then x° is **a**.
(2) if x is a number in M, say $x = \langle 0, n\rangle$, then $x^\circ = \bar{n}$.
(3) if c^* is an object in M, where c is a constant of **ZFR**, then $c^{*\circ}$ is c.
(4) if x is in the first or third column below, then x° is in the next column:

$\langle 1,0,x\rangle$	$\mathbf{k}x^\circ$,	$\langle 1,4,\langle 0,n\rangle,\langle 0,m\rangle\rangle$	$\mathbf{d}\overline{\overline{nm}}$,
$\langle 1,1,x\rangle$	$\mathbf{s}x^\circ$,	$\langle 1,4,\langle 0,n\rangle,\langle 0,m\rangle,x\rangle$	$\mathbf{d}\underline{nm}x^\circ$,
$\langle 1,1,x,y\rangle$	$\mathbf{s}x^\circ y^\circ$,	$\langle 1,6,'\phi',a\rangle$	$\mathbf{c}_\phi\mathbf{a}$,
$\langle 1,2,x\rangle$	$\mathbf{p}x^\circ$,	$(1,6,'\phi',a,y_1,\ldots,y_k\rangle$	$\mathbf{c}_\phi(\mathbf{a},y_1^\circ,\ldots,y_k^\circ)$,
$\langle 1,4,\langle 0,n\rangle\rangle$	$\mathbf{d}\bar{n}$,	$\langle 1,7,a\rangle$	**im a**.

11.7. Theorem. *The map ° is a homomorphism of combinatory algebras, i.e., it preserves application.*

Proof. The statement that it is a homomorphism also means that $\mathbf{k}_M$ goes onto **k** and $\mathbf{s}_M$ goes onto **s**, which is true by definition of °. We prove that it preserves application. We shall prove by induction on the definition of the application relation in M that if $App(x,y,z)$, then $RED(x^\circ y^\circ) = z^\circ$, where, as above, RED is call-by-value term reduction. There are ten cases corresponding to the clauses (1)–(10) of 11.3.

Case 1. We have $App(\mathbf{k}^*, x, \langle 1,0,x\rangle)$; we have to show that A satisfies

$$RED(\mathbf{k}x^\circ) = (\langle 1,0,x\rangle)^\circ.$$

Both sides of this equation are equal to $\mathbf{k}x^\circ$. For the second part of (1), we have $App(\langle 1,0,x\rangle, y, x)$. Note that $(\langle 1,0,x\rangle)^\circ = \mathbf{k}x^\circ$, so that what we must show is $RED(\mathbf{k}x^\circ y^\circ) = x^\circ$; but this is true since x° and y° are normal terms.

Case 2. We have $App(\mathbf{s}^*, x, \langle 1,1,x\rangle)$; we have to show that A satisfies

$$RED(\mathbf{s}x^\circ) = (\langle 1,1,x\rangle)^\circ;$$

but both sides of this equation are equal to $\mathbf{s}x^\circ$.

Case 3. Suppose we have $App(x, z, u) \wedge App(y, z, v) \wedge App(u, v, w)$. Then, by induction hypothesis, we have

$$RED(x^\circ z^\circ) = u^\circ, \qquad RED(y^\circ z^\circ) = v^\circ, \qquad RED(u^\circ v^\circ) = w^\circ. \qquad (1, 2, 3)$$

We must show $RED((\langle 1, 1, x, y\rangle)^\circ z^\circ) = w^\circ$. We have $(\langle 1, 1, x, y\rangle)^\circ = \mathbf{s}x^\circ y^\circ$; now we calculate

$$\begin{aligned} RED(\mathbf{s}x^\circ y^\circ z^\circ) &= RED(RED(x^\circ z^\circ) RED(y^\circ z^\circ)) \\ &= RED(u^\circ v^\circ) \quad \text{by equations (1) and (2)} \\ &= w^\circ, \end{aligned}$$

which was what we had to show.

Case 4. We have $App(\mathbf{p}^*, x, \langle 1, 2, x\rangle)$; we have to show that A satisfies

$$RED(\mathbf{p}x^\circ) = (\langle 1, 2, x\rangle)^\circ;$$

but both sides of this equation are equal to $\mathbf{p}x^\circ$. For the second part of (4), we have $App(\langle 1, 2, x\rangle, y, \langle 2, \{x, y\}\rangle)$. Note that $(\langle 1, 2, x\rangle)^\circ = \mathbf{p}x^\circ$, so that what we must show is $RED(\mathbf{p}x^\circ y^\circ) = (\langle 2, \{x, y\}\rangle)^\circ$. The right-hand side is equal to $\mathbf{a}$ by definition of $^\circ$. The inner terms on the left-hand side, namely x° and y°, are normal; so the left-hand side is computed using the reduction rule $\mathbf{p}xy$ reduces to $\mathbf{a}$.

Case 5. We have $App(\mathbf{u}^*, x, \langle 2, w\rangle)$ for a certain w (the exact value of w is given just before Definition 11.3, but does not matter); since $(\langle 2, w\rangle)^\circ = \mathbf{a}$, what we must show is $RED(\mathbf{u}x^\circ) = \mathbf{a}$. Since x° is normal, this follows using the reduction rule $\mathbf{u}t$ reduces to $\mathbf{a}$.

Case 6. First assume ϕ has only x free. Then we have $App(\mathbf{c}^*_\phi, \langle 2, a\rangle, \langle 2, \{w \in a : \phi^*(w)\}\rangle)$. We have to show $RED(\mathbf{c}_\phi\{2, a\}^\circ) = \{2, \{w \in a : \phi^*(w)\}\}^\circ$. But both sides are equal to $\mathbf{a}$. Next assume that ϕ has exactly $x, y_1, \ldots, y_m$ free. Then we have $App(\mathbf{c}^*_\phi, \langle 2, a\rangle, \langle 1, 6, '\phi', a\rangle)$. We must show $RED(\mathbf{c}_\phi \mathbf{a}) = \langle 1, 6, '\phi', a\rangle^\circ$. The left-hand side is equal to $\mathbf{c}_\phi\mathbf{a}$ since this term is normal (because $m > 0$), and the right-hand side is equal to $\mathbf{c}_\phi\mathbf{a}$ by definition of $^\circ$.

Similarly, we have $App(\langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_k\rangle, \quad y_{k+1}, \quad \langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_{k+1}\rangle$. We must show that

$$RED(\mathbf{c}_\phi \mathbf{a} y_1^\circ, \ldots, y_{k+1}^\circ) = \langle\langle 2, a\rangle, y_1, \ldots, y_{k+1}\rangle^\circ.$$

The term on the left side is normal, and by definition of $^\circ$, it is the value of the right-hand side also. We have

$$App(\langle 1, 6, '\phi', \langle 2, a\rangle, y_1, \ldots, y_{m-1}\rangle, y_m, \langle 2, \{x \in a : \phi^*(x, y_1, \ldots, y_m)\}\rangle).$$

We must show $RED(\mathbf{c}_\phi \mathbf{a} y_1^\circ, \ldots, y_m^\circ) = \mathbf{a}$. This follows from the fact that the subterms on the left are normal, together with one of the reduction rules.

Case 7. We have $App(\mathbf{s}^*_N, \langle 0, n\rangle, \langle 0, n \cup \{n\}\rangle)$. We must show $RED(\mathbf{s}_N \bar{n}) = \overline{n+1}$. Since $\overline{n+1}$ is just another name for the term $\mathbf{s}_N\bar{n}$, this follows from the fact that numerals are normal terms.

Case 8. We have $App(\mathscr{P}^*, \langle 2, x\rangle, \langle 2, \{\langle 2, z\rangle : z \in P(x)\}\rangle)$. We must show $RED(\mathscr{P}\mathbf{a}) = \mathbf{a}$, which follows since $\mathscr{P}$ and $\mathbf{a}$ are normal and $\mathscr{P}\mathbf{a}$ reduces to $\mathbf{a}$.

Case 9. We have $N(n)^* \to App(\mathbf{d}^*, n, \langle 1, 4, n\rangle)$. Suppose $N(n)^*$, that is, n is an integer in M. Then n has the form $\langle 0, n_1\rangle$, and n° is $\bar{n}_1$. So what we have to show is $RED(\mathbf{d}\bar{n}_1) = (\langle 1, 4, n\rangle^\circ)$. Since $\mathbf{d}\bar{n}$ is normal, the left side is just $\mathbf{d}\bar{n}_1$. That is also the value of the right-side, by definition of $^\circ$.

We have $N(n)^* \wedge N(m)^* \to App(\langle 1, 4, n\rangle, m, \langle 1, 4, n, m\rangle)$. Suppose $N(n)^*$ and $N(m)^*$. Then n has the form $\langle 0, n_1\rangle$, and m has the form $\langle 0, m_1\rangle$. What we have to show is $RED(\mathbf{d}\bar{n}_1\bar{m}_1) = (\langle 1, 4, n, m\rangle)^\circ$. Since the term on the left is normal, the left side is just $\mathbf{d}\bar{n}_1\bar{m}_1$. By definition of $^\circ$, that is also the value of the right side.

We have $N(n)^* \wedge N(m)^* \to App(\langle 1, 4, n, m\rangle, x, \langle 1, 4, n, m, x\rangle)$. Suppose, as above, that $n = \langle 0, n_1\rangle$ and $m = \langle 0, m_1\rangle$. We have to show $RED(\mathbf{d}\bar{n}_1\bar{m}_1 x^\circ) = (\langle 1, 4, n, m, x\rangle)^\circ$. Since x° is normal, the left side is just $\mathbf{d}\bar{n}_1\bar{m}_1 x^\circ$. But that is the value of the right side, by definition of $^\circ$.

Suppose $n = \langle 0, n_1\rangle$. We have $App(\langle 1, 4, n, m, x\rangle, y, x)$. We have to show $RED(\mathbf{d}\bar{n}_1\bar{n}_1 x^\circ y^\circ) = x^\circ$. But that follows from the facts that x°, y°, and numerals are normal terms, so the reduction rule for $\mathbf{d}$ applies.

Similarly, suppose $n = \langle 0, n_1\rangle$ and $m = \langle 0, m_1\rangle$. Then we have $App(\langle 1, 4, n, m, x\rangle, y, y)$. We have to show $RED(\mathbf{d}\bar{n}_1\bar{m}_1 x^\circ y^\circ) = y^\circ$. But that follows from the facts that x°, y°, and numerals are normal terms, so the reduction rule for $\mathbf{d}$ applies.

Case 10. Suppose b is a set in M. Then we have $App(\mathbf{im}^*, b, \langle 1, 7, b\rangle)$. We have to show that $RED(\mathbf{im}\, b^\circ) = \langle 1, 7, b\rangle^\circ$. The right side is $\mathbf{im}\,\mathbf{a}$, by definition of $^\circ$. The left side is the same since $b^\circ = \mathbf{a}$ and $\mathbf{im}\,\mathbf{a}$ is a normal term. Now suppose that b is a set in M. Suppose furthermore that M satisfies

$$\forall x \in_M b \exists y \in_M u \mathbf{App}(f, x, y) \wedge \forall y \in_M u \exists x \in_M b \mathbf{App}(f, x, y).$$

Then we have $App(\langle 1, 7, b\rangle, f, u)$. Note that $\langle 1, 7, b\rangle^\circ = \mathbf{im}\,\mathbf{a}$ since $b^\circ = \mathbf{a}$. We must show

$$RED(\mathbf{im}(\mathbf{a}, f)^\circ) = u^\circ.$$

Since f° is normal, the reduction rule $\mathbf{im}(\mathbf{a}, x) = \mathbf{a}$ applies, so the left side is just $\mathbf{a}$. Since u is a set in M, u° is also $\mathbf{a}$. That completes the ten cases of the inductive proof. □

11.8. Theorem. *The model M satisfies classical logic and also Church's thesis in the form, every operation from* N *to* N *is recursive.*

Proof. Suppose f is an element of M such that M satisfies $\forall n \in \mathbf{N}(fn \in \mathbf{N})$. Let t be f°. By Theorem 11.7, if M satisfies $f\bar{n} = \bar{m}$, then $t\bar{n}$ reduces to $\bar{m}$ by call-by-value reduction. In other words, f is extensionally equal to its image f°. But by Lemma 11.5, this function is recursive. □

Another interesting consequence can be drawn from the model M: it is impossible to define an operation in **ZFR** which will take two distinct integer values on two different sets a and b. Before proving this fact, we first make it plausible. Suppose a and b are two sets; then define $u = \{x : (x \in a \wedge P) \vee (x \in b \wedge \neg P)\}$, where P is an unsolved problem, say Fermat's last theorem. Now suppose we have an operation f such that $fa = 0$ and $fb = 1$. Then we could settle Fermat's problem by computing fu and seeing whether it is zero or one. This kind of example is familiar in constructive mathematics, but here we have classical logic; only the operations are to be constructive.

11.9. Theorem. *Suppose a and b are two sets in M. Suppose f is an element of M such that M satisfies $fa \in \mathbf{N}$ and $fb \in M$. Then M satisfies $fa = fb$.*

Proof. Let m be the value of fa in M and n the value of fb. Then $f^\circ(\mathbf{a})$ reduces to $\bar{m}$ since $^\circ$ is a homomorphism and $a^\circ = \mathbf{a}$. But since also $b^\circ = \mathbf{a}$, $f^\circ(\mathbf{a})$ reduces to $\bar{n}$. By the uniqueness of normal forms, $n = m$. □

In particular, any integer-valued operations defined on $\mathcal{P}(\mathbf{N})$ must be constant. It is impossible to define nontrivial *integer-valued* operations on sets. It may seem paradoxical to propose a computation system based on set theory when such a simple function as $count(x)$, the cardinality of a finite set x, cannot be computable. But as we have seen if $count(x)$ were computable, then we could solve Fermat's last theorem, by applying *count* to a suitable set. That set is not the sort of set we will deal with often in computer science since we cannot list its elements. There are two ways of saying what kinds of sets we will deal with in computer science: either use constructive logic, in which case "finite set" has a more restrictive meaning than it does in classical logic; it means we can explicitly count the elements of the set. Another approach is to restrict attention to *representations* of sets, e.g., by finite lists, which are operations with domain an initial segment of **N**. In any case, our inability to count the number of elements of sets with complicated definitions will not be a practical limitation on the usefulness of set theory as a computation system.

Remark. It is an open problem whether Church's thesis plus classical logic plus **Exp** is consistent.

12. IZFR as a computation system

We now return to the original theme of the paper, namely set theory as a computation system. We shall discuss the possibility of implementing **IZFR** as a computation system. The first step is to write the axioms of **IZFR** as rules of inference in a natural deduction system. Such a system shows how to infer lines of the form $A : \Gamma$ from other such lines, where Γ is a list of assumptions on which the formula

A depends. The axioms of **IZFR** can be viewed as type-formation rules, e.g.,

$$\frac{S(a)}{S(\mathbf{c}_\phi(a))}$$

together with introduction rules like

$$\frac{x \in a \qquad \phi(x)}{x \in \mathbf{c}_\phi(a)}$$

and the elimination rules like

$$\frac{x \in \mathbf{c}_\phi(a)}{x \in a}, \qquad \frac{x \in \mathbf{c}_\phi(a)}{\phi(x)}.$$

Note that the axiom of images can also be treated this way, while there is no (natural) corresponding version of the collection axiom:

$$\frac{S(a) \qquad S(fx)\colon x \in a}{S(\mathbf{im}(a,f))},$$

$$\frac{S(a) \qquad S(fx)\colon x \in a \qquad z \in a}{fz \in \mathbf{im}(a,f)},$$

$$\frac{S(a) \qquad S(fx)\colon x \in a \qquad z \in \mathbf{im}(a,f)}{\exists x \in a(z = fx)}.$$

These rules can be straightforwardly translated into PROLOG, producing a computation system that recognizes or finds proofs in **IZFR**. Of course, such a system could be implemented in any other programming language, too.

What would such a computation system be good for? Consider Knuth's example of an algorithm extracted from Bishop's proof of the Weierstrass approximation theorem. We could write out the formal proof of the approximation theorem, feed it to a suitable (rather simple) program, and extract the algorithm implicit in the proof. We should be able, in fact, to use the computation system to fill in the most detailed steps of the proof if we provide the major steps. In other words, we should be able to extract algorithms from proofs; such algorithms would be automatically guaranteed to satisfy their specifications.

An actual implementation would probably avoid existentially defined sets such as $\mathbf{im}(a,f)$ and work instead with a term for the graph of f restricted to a; in this paper we have used **im** instead, in the belief it is more natural for humans. The computer, however, would have to keep track of the "witnesses" demonstrating membership in $\mathbf{im}(a,f)$, and it would be natural to do this explicitly. Such an implementation would probably restrict the separation axiom to formulae which are "almost-negative" in a suitable sense. This would permit the use of a simple form of q-realizability for the extraction of algorithms from proofs. On top of this internal system there would be a user interface which would translate certain existentially defined types into explicit, internal versions. Thus, for example, the

user could define a type as a union, and the system would construct an internal type which would also keep track of the witness of membership in the union.

We next discuss some examples of data types which are useful in computer science, as defined in such a computerized set theory.

Example 1 (*Lists*). The programming language ML permits the polymorphic type α *list*, where α is any type; this is the type of lists of objects of type α. In **ZFR** this can be defined in the most mathematically natural way: a list of length n is a function from the first n integers to the set α, and $list(\alpha)$ is the union over all integers n of the set of lists of length n of members of α.

Example 2 (*Sorting*). One wishes to be able to define a polymorphic sorting algorithm $sort(\alpha, R, x)$ which will take as arguments any set α with a linear order R on it, and a finite list x of elements of α, and return the sorted list with the same elements as x, but in R-order. Here we assume that R is actually an operation that decides the order of any two elements of α by returning 1 or 0. Several such algorithms are known and studied in elementary courses, yet many computer languages, even such advanced ones as ML, are unable to express such a polymorphic operation. Instead, you must write a new version of your sorting algorithm for each new type α. (The reason is that $sort(\alpha, R, x)$ is only defined when R satisfies a condition depending of α, and the type-formation mechanisms of ML do not allow for that.) Note that definitions of this sort are completely straightforward in **ZFR**.

13. Church's thesis and the axiom of choice

The first model constructed above contains lots of nonrecursive functions from N to N. It therefore does not correspond to any sensible operational semantics for set theory considered as a computation system. We therefore constructed the second model M, in which only recursive operations occur, even if logic is taken to be classical. If we do take classical logic, then exactly as in ordinary set theory there will be many nonrecursive *functions*, in the set-theoretic sense of single-valued subsets of $\mathbf{N}\times\mathbf{N}$. But the natural implementation of **IZFR** as a computation will make use of only the intuitionistic natural-deduction rules; the law of the excluded middle is an extra appendage in this context.

Such considerations lead naturally to the formal question whether **IZFR** can prove the existence of any nonrecursive functions. In other words, is it consistent with **IZFR** to assume that all functions are recursive? Note that this is stronger than "all operations are recursive"; we already know that this is consistent, by the model M of Section 11.

Church's thesis can be formulated in the language of **IZF** as follows:

(**CT**) $\forall f \in \mathbf{N}^{\mathbf{N}} \exists e \in \mathbf{N}(e \text{ is a recursive index of } f)$.

13.1. Theorem (Consistency of Church's thesis). *It is consistent with* **IZFR** *to assume all functions are recursive. What is more, every theorem of* **IZFR+CT** *in the language of* **IZF** *is already provable in* **IZF+CT**.

Proof. It is known (see, e.g., [4, Chapter VIII]), that **IZF** is consistent with **CT**. Now suppose that **IZFR+CT** proves ϕ. Then, by the deduction theorem, **IZFR** proves $\mathbf{CT} \to \phi$. By Theorem 10.5, **IZF** proves ϕ, assuming ϕ is in the language of **IZF**. □

The principle of *countable choice* can be naturally expressed in **ZFR** by

$$(\mathbf{AC_N}) \quad \forall n(N(n) \to \exists x A(x, y)) \to \exists f \forall n(N(n) \to A(n, fn)).$$

Note that $\mathbf{AC_N}$ can be used to prove that every that every set-theoretic function on **N** is extensionally equivalent to some operation. Similarly, the following axiom of "unique choice" can be used to prove that every set-theoretic function (with whatever domain) is extensionally equivalent to some operation:

$$\forall x \in a \exists ! y A(x, y) \to \exists f \forall x \in a A(x, fx).$$

This axiom was first introduced by Myhill [41] who called it an axiom of "non-choice" since if f ranges over set-theoretic functions instead of operations, it is a triviality. Both the axioms of choice just mentioned are consistent with **ZFR** since they hold in the model constructed in Section 9. On the other hand, they fail in the model M of Section 11 since they imply **FO**, which fails in that model since there are nonrecursive functions but only recursive operations. It is therefore of interest to prove the following theorem.

13.2. Theorem. $\mathbf{IZFR+CT+AC_N+AC!}$ *is consistent.*

Proof. **AC!** is a consequence of **FO**, so it suffices to prove that $\mathbf{IZFR+FO+AC_N}$ is consistent. Suppose it is inconsistent. Then some finite conjunction B of instances of $\mathbf{AC_N}$ can be refuted in **IZFR+FO**. Note that using **FO**, the operation mentioned in $\mathbf{AC_N}$ can be replaced by a function, so that the formula B can be assumed to be a formula of **IZF**, i.e., not mentioning **Ap**. By Theorem 10.5, then B is refutable in **IZF-Rep**. But B is a finite conjunction of instances of countable choice in the version expressible in **IZF**, i.e., with functions instead of operations. Hence the refutation of B in **IZF** contradicts the known consistency of **IZF** with countable choice. (See, for example, [4, Chapter VIII].) □

Appendix. The concepts of set, class, and data type

The ideas of constructive mathematics and computer science prompt a re-examination of the concepts of "set" and "class". In this appendix, we shall recall the ideas of Cantor, Dedekind and Frege, and compare them with those of Brouwer

and Bishop, and with the "data types" of computer science, as explained by Martin-Löf and as used in modern programming languages.

What is a set?

Cantor addressed this fundamental question at the beginning of his seminal memoir (1895) on set theory:

> A set is a collection into a whole of definite, distinct objects of our intuition or thought.[4]

Cantor was well aware that there are certain "inconsistent sets", as he called them, which he regarded as "multiplicities" that could not be collected into "unities". Thus the phrase "into a whole" was vital.

Another person who was influential in the development of set theory was Dedekind, who used set theory to give a foundation to arithmetic in his essay, *Was sind und was sollen die Zahlen.* Here is his definition of a set (for which he used the word "system"):

> If different things $a, b, c, \ldots$ for some reason can be considered from a certain point of view, can be associated in the mind, we say they form a *system* $S \ldots$ Such a system (an aggregate, a manifold, a totality) as an object of our thought is likewise a thing; it is completely determined when with respect to everything it is determined whether it is an element of S or not.[5]

This definition, which nicely rules out the set occurring in Russell's paradox, preceded Russell's discovery of that paradox by more than two decades. In both Cantor's and Dedekind's definitions, there is an element of "mental collection" which must be present in order that elements constitute a set. Nevertheless, it seems to have been the intention of both that a set was determined by its elements. The very next sentence in Dedekind's book makes this explicit, by formulating what is now known as the axiom of extensionality:

> The system S is hence the same as the system T, in symbols $S = T$, when every element of S is also an element of T, and every element of T is also an element of S.

Independently of Dedekind and Cantor, Frege was developing his own theory of classes. His theory was based on the idea of a "concept" or property which an object might or might not have, and permitted the formation of what we would not write as $\{x : \phi(x)\}$ for each concept ϕ. ϕ was viewed as what today we would call a "Boolean-valued function", associating to each x a truth-value "the True" or "the False". In the introduction to his *Grundgesetze* (1893) [17, p. 149], he criticizes Dedekind's definition of set:

> ... but the "considering", "putting together in the mind", is not an objective characteristic. In whose mind, may I ask? If they are put together in one mind and not in another, do they then form a system? What is to be put together in my mind must doubtless be in my mind. Then do things outside myself not form systems? Is a system a subjective formation in each single mind? Is then the constellation Orion a system?

[4] "Unter einer 'Menge' verstehen wir, jede Zusamenfassung M unserer Anschauung oder unseres Denkens zu einem Ganzen." The translation given in the text is Fraenkel's [16, p. 9].

[5] Dedekind [3, p. 45]. The original date of publication was 1887; however, the quotation is from the second edition, published in 1893.

Frege goes on to the example of the empty set, making the point that if it is the elements that determine a set, then it is hard to see how a set with no elements can be determined! Instead, he points out, the empty set is the set of all things falling under a concept which is always false.[6] The intuitive appeal of this theory of concepts and classes was such that Frege was convinced of its value. He wrote [17, p. 147]:

> It is improbable that such an edifice could be erected on an unsound base. Those who have other convictions have only to try to erect a similar construction upon them, and they will soon be convinced that it is not possible, or at least it is not easy. As a proof of the contrary, I can only admit the production by some one of an actual demonstration that upon other fundamental convictions a better and more durable edifice can be erected, or the demonstration by some one that my premises lead to manifestly false conclusions. But nobody will be able to do that.

Of course, this confidence was soon shattered by Russell's paradox. The situation after this paradox had been fully apprehended is aptly summarized by Zermelo (1908) [40], [26, p. 200]:

> Cantors original definition of a set... therefore certainly requires some restriction; it has not, however, been successfully replaced by one that is just as simple and does not give rise to such reservations. Under these circumstances there is at this point nothing left for us to do but to proceed in the opposite direction, and, starting from set theory as it is historically given, to seek out the principles required for establishing the foundations of this mathematical discipline. In solving the problem we must, on the one hand, restrict these principles sufficiently to exclude all contradictions and, on the other, take them sufficiently wide to retain all that is valuable in this theory.

This ad hoc procedure, which after initial controversies became the dominant view of the mathematical community, and remains so to this day, is philosophically unsatisfying. Not everyone followed this path; in particular Whitehead and Russell attempted to repair Frege's logical foundations by introducing the theory of types in *Principia Mathematica*.[7]

Both Russell-Whitehead and Zermelo gave formal systems which avoided the paradoxes while allowing ordinary mathematics, but their motivations differed: Zermelo looked to history and practice for his axioms, Russell and Whitehead looked for intuitively correct *logical* notions. At that time, the theory of classes was often viewed as logical in nature; even Cantor had considered such a deep mathematical proposition as the well-ordering theorem as a "law of thought". So it is not surprising that the paradoxes gave rise to a reexamination of the laws of logic. Such a "beginning again at the beginning" was undertaken by Brouwer.[8] In 1918, in

[6] The empty set is nowhere to be found in Cantor, and is specifically eschewed by Dedekind for "certain reasons" which are left unspecified. So perhaps they were aware of this difficulty!

[7] The closeness of viewpoint in *Principia* and in Frege can be seen for example in *Principia* *20: "The characteristics of a class are that it consists of all the terms satisfying some propositional function, so that every propositional function determines a class, and two functions which are formally equivalent (i.e., such that whenever one is true, the other is true also) determine the same class, while conversely two functions which determine the same class are formally equivalent."

[8] It is worth noting that merely rejecting the law of the excluded middle does not stop Russell's paradox, which can be carried out with intuitionistic logic. Frege seems to have been confused about this [17, p. 325].

Begründung der Mengenlehre unabhängig vom logischen Satz vom ausgeschlossenen Dritten (*Foundations of set theory independent of the logical theorem of the excluded middle*) [7], Brouwer addresses the problem of defining the fundamental notion of "set". According to Brouwer, the natural numbers and the linear continuum are known to us by intuition; thus natural numbers and real numbers (given by what later came to be called "choice sequences") will be the elements of the most fundamental kind of set. Brouwer defined a set ("Menge") or *spread* to be (ignoring some minor technicalities) a *rule* associating a number to each sequence of natural numbers. "*Eine Menge ist ein Gesetz*..."; a set is a rule. More generally, Brouwer recognized the notion of a "species of first order", whose members could be only numbers or spreads, and which had to be defined by a property. Quite possibly influenced by Whitehead and Russell's theory of types, he went on to recognize "species of second order" whose elements could be species of first order, and so on.

Brouwer's approach was felt to be too radical by many mathematicians. For example, Von Neumann (1925) [38], in introducing a new axiomatization of set theory, is at pains to separate his and Zermelo's approach from that of Brouwer, which is "not a rehabilitation of set theory at all, but rather a very sharp critique of the modes of inference hitherto used in elementary logic... [Brouwer] systematically rejects the larger part of mathematics and set theory as completely meaningless." That this was, for Von Neumann, enough to vitiate the entire approach can be seen from these remarks:

> There will be no attempt to make derivations unobjectionable also in the sense of the intuitionism of Brouwer and Weyl. I would like to remark, nevertheless, that this, too, could be attained rather easily (through a few insignficant modifications); but I forgo this as a matter of principle, since the axiomatic method is in itself contrary to the essence of intuitionism. [26, p. 396].

Constructive axiomatic set theory thus missed a chance of being born in 1925; it had to wait almost half a century more. Brouwer's intuitionistic set theory gained few followers, no doubt because of its wholesale rejection of classical concepts and methods. So far as the author is aware, nobody tried again to define the concept of "set" until Bishop (1967) [5]. There one finds the following definition (p. 13):

> The totality of all mathematical objects constructed in accordance with certain requirements if called a *set*.

The word "requirements" reminds one of Frege's "concepts"; but note the explicit provision that the elements must be constructed. The word "totality" reminds one of Cantor and Dedekind (in fact, it was given as a synonym for "system" by Dedekind); but the subjective nature of the process of "collecting" into a totality, which Frege criticized, is not present in Bishop's definition. Bishop seems to have believed, or at least hoped, that he had done what Zermelo mentions above as not having been done: to successfully replace Cantor's definition of set by one just as simple which does not give rise to paradoxes. In unpublished work, Bishop gave a quite attractive axiomatic theory in which his book could be formalized; but,

unfortunately, like Frege's, it was inconsistent. (Presumably he was aware of the fact, which was why he did not publish his theory.)[9]

Bishop's explanation of "set", however, is not at all what is taught in mathematics courses in graduate school. If the question of the nature of sets is taken up at all, the explanation offered is the "cumulative hierarchy" of sets. One starts with the empty set (or some individuals) and repeatedly collects the subsets of what one has so far:

$$R_\alpha = \bigcup \{R_\beta : \beta < \alpha\}.$$

It is said that this is the "intended model" of the axioms of set theory. It is interesting that this "standard model" was not discussed at the time when ZF was created. The axiom of foundation was added to ZF in 1928 by Von Neumann, who also introduced the cumulative hierarchy (although that was presaged by Mirimanoff in 1917 [35]). ZF was not about any specific notion of set, such as the well-founded sets, when it was first given; it was only hoped that the paradoxes would be avoided.

We thus have (at least) two distinct ideas of "set": the Fregean notion of a set as an extension of a property, which is intuitively appealing but inconsistent, and the notion of well-founded set, which is *defined* in terms of the (presumably) more primitive notions of iteration and collection of subsets. Whether we adopt a constructive philosophy or not seems to make little difference: the cumulative hierarchy can be treated, at least formally, with intuitionistic logic (Friedman [18]), and in spite of hopes to the contrary, constructive set theory does no better at avoiding the paradoxes than classical set theory.[10]

What is a type?

By a "type" we mean what a computer scientist means by "data type". These tell "what kind of an object" a given piece of data represents. In modern structured languages like Pascal, a program begins with "type declarations" in which certain variable names (identifiers) are reserved to name objects of the declared types. The history of the development of programming languages shows the continual search for more flexible and general types. FORTRAN had only two types: real and integer. Pascal allows more general type-forming operations, but still does not have a type $A \to B$ of operations from the type A to the type B. In the future, languages will be created which are adequate both to the needs of programmers and to the expression of mathematics. A pioneering effort in this direction was the AUTOMATH project (see, e.g., [6]); some others have been surveyed in Section 4.

[9] For completeness one should note that Bishop felt that every set should be equipped with a relation of equality, and that one has not completely defined a set until one has also defined its equality relation. Although this is not part of the "definition" quoted, it occurs two sentences further on.

[10] Some may argue that Bishop's definition constitutes a third "distinct idea" since it cannot easily be equated with either Frege's notion or with Von Neumann's. Whether or not we so count it will not be essential to what follows.

By "data" we understand concrete objects which can be directly and completely represented in the computer; for example, strings of symbols. Data are to be contrasted with the abstract objects of mathematics, for example, infinite sets, which can be only indirectly represented in the computer, for example, by a program which decides membership in the set, or a string of symbols which defines the set in some language.

As we shall use the terms, the difference between a type and a set is this: a type may have only concrete objects as its members, while a set may have concrete or abstract objects as its members. In particular, a type is itself not a concrete object (unless it is a finite type). In computer science, the idea of a hierarchy of types does not arise. One specifies types by specifying what objects "have that type", i.e., are members of the type. That is, types are typically defined in Fregean terms as extensions of a concept.

Martin-Löf [33, 34] develops a "theory of types" which permits more general type-forming operations than Pascal; in particular, the type-forming operation $A \to B$ is allowed, and its generalization to the type-formation operation $\Pi(x \in A)B(x)$, where $B(x)$ is a type for each $x \in A$. ($A \to B$ is the special case when $B(x)$ is constant.) These type theories, as published, permit the creation of a hierarchy of types; there is a "small universe" U containing the basic types, and closed under the type-forming operations; U is itself a type. So the principle that types cannot be objects is violated. In unpublished lectures (but the formalism appears in Smith [42]), Martin-Löf rectifies this situation by emphasizing that only *names* of types can be objects. He introduces names for all the small types and makes *these* the elements of U.

Martin-Löf gives an explanation of what he means by a "type"; his formal theories are supposed to reflect this informal notion. According to Martin-Löf, in order to specify a type X, you must tell

(i) what the canonical objects of type X are

(ii) when two canonical objects of type X are equal, and

(iii) how to reduce an arbitrary object of type X to canonical form.

For example, the canonical members of the type of integers are given by: 0 is a canonical element, and if n is a canonical element, so is $s_N n$. Whenever we introduce an operation on the integers, such as addition or multiplication, we have to give the corresponding rules for evaluating expressions involving that operation. Thus 10^{10} is an integer, because there are rules for evaluating that expression.

Let us compare Martin-Löf's types with Bishops sets. According to Bishop, we have given a set X as soon as we have told what to do to construct an element of X, and when two elements of X are equal. To turn X into one of Martin-Löf's types, we still have to tell what the canonical elements are and how to reduce arbitrary elements to canonical form.[11] Thus, for example, the set of all sets would appear to qualify as a Bishop set since he has just told us what to do to construct

[11] Martin-Löf ignores the logical distinction between elements and expressions denoting elements, and speaks of reducing elements where a logician would insist that only expressions are reduced.

a set.[12] But it does not qualify as a data type since we do not have any idea what the canonical sets are, nor what reduction operations are appropriate.

Feferman [14, 15] introduces theories of operations and "classes" which are intended to express (and be obviously valid on) a philosophy according to which mathematical objects are given by symbolic representations (i.e., data). His formalism permits classes to be objects, but it is easy to modify the formalism (see [3]) to permit only *names* of classes to be objects, without affecting any important properties of the system. In particular, the key axiom of Feferman's systems is the "elementary comprehension axiom", according to which we are allowed to form the class $\{x : \phi(x)\}$ for certain ϕ. One thereby has the existence of a "universe" V. The ϕ which are allowed are called "elementary"; they are distinguished by the syntactic property that class variables occur only free (as parameters) and on the right of $\in$ in ϕ. This corresponds to the idea that types can be mentioned while defining new types only if they have been previously defined (so they can be substituted for the free class parameters in ϕ). Allowing them only on the right of $\in$ corresponds to the idea that they are not allowed to be objects themselves. The Russell paradox is then neatly avoided since $x \notin x$ is not an elementary formula.

Graves [22] is developing a computation system based on category theory (more precisely, on topos theory). He considers the category whose objects are data types; the arrows of the category are procedures leading from inputs of the domain type to outputs of the codomain type. One object, the terminal object of the category, is the type *ONE*. *ONE* is thought of as the type of "environments", i.e., maps which give values to variables. A data object is an arrow of type $ONE \to X$. We would usually call this an object of type X. For instance, 5 is the map of type $ONE \to \mathbf{N}$ with constant value 5. The computations in his system consist in the reductions or evaluations of terms in topos theory (enriched by a suitable collection of function symbols). Graves' work shows how topos theory can be used as a foundation for the theory of data structures and algorithms, although it was originally developed as an alternate foundation for mathematics.

One may ask: from the philosophical point of view, what does topos theory contribute to our understanding of the fundamental notion of data type? One answer is, it provides an operational semantics. A notion of data type is sound if the topos-theoretic formalism can define it; or perhaps, if a natural extension of the formalism can define it. Graves has shown that all the usual data types can be naturally defined in topos theory, which of course goes far beyond Pascal in its generality of type formation.[13]

[12] Greenleaf [23, pp. 224–225], argues against this, on the grounds that the set of all sets does not qualify as a Bishop set until we have constructed an equality relation on it, which he thinks is not so easy.

[13] There is, however, a technical problem suggested by this definition of soundness which is presently still open: how can we interpret Martin-Löf's type-construction operations in topos theory?

Classes

The word "set" can be used to refer either to a Fregean set, defined by the extension of a property, or to a well-founded set, i.e., one occurring in the cumulative hierarchy of sets. The word "class" refers to the extension of a property. If a class happens to be a well-founded set, then it is usually referred to as a set. Thus the word "class" is sometimes thought to be synonymous with "proper class", which refers to a class which is not in the cumulative hierarchy, such as the class of all well-founded sets or the class of all ordinals.

In the history of set theory, proper classes go back to Cantor, who recognized that certain "multiplicities" cannot be collected into a "unity". He referred to such multiplicities as "inconsistent sets". It seems to have been Von Neumann [38] who first contemplated the consistent use of proper classes, though his theory is formulated in terms of functions rather than sets and classes. Later formulations, more familiar today, were given by Bernays and finally by Gödel [43]. All these theories regard classes as being just like sets except that they are, in Von Neumann's picturesque terminology, "too big". That is, "too big" to be a member of another class. In particular, they are viewed as abstract objects like sets, obeying the law of extensionality: two classes are equal if and only if they have the same members.

This viewpoint is not philosophically sound. If a class is "too big" to be a set, that means it cannot be collected into a unity *as an abstract object.* The process of abstraction that leads to a set from a symbolic representation *breaks down* on certain symbolic representations. A class is given to us by a symbolic representation, but it has no extension. Frege's "second-level function" of abstraction that leads from properties to their extensions is not everywhere defined. (See [17, p. 239] for Frege's analysis of Russell's paradox in exactly these terms.) A class, then, is a concrete object, not an abstract one. It is the *extension* of a class which is "too big" to exist; or rather, *would be* too big if it *did* exist.

It follows that there is no justification for the principle of extensionality for classes, in spite of its formal consistency. Nor should we identify a class with the set, if it exists, which has the same members: one is a concrete object, the other is an abstraction.

The way in which data types are used in computer science casts some light on the situation. A type can be used in two different ways:

(i) as a *classifier*, for classifying objects according to what type they are, or

(ii) as a data structure, i.e., an object, in its own right, which can be manipulated like any other piece of data.

Data types as classifiers are like sets; data types as structures are like classes. Since this is rather a revolutionary view of the matter, perhaps we had better put it the other way around: sets are like data types as classifiers, while classes are like data types as structures.

Some support for this view comes, quite independently of the ideas that led to it, from the work of Peter Aczel. Aczel has been trying to use set theory to describe

certain data structures used in computer science, for example *streams*. A stream is a (theoretically) infinite object, for example the stream of characters being printed on a terminal. Aczel [1] found that the well-founded sets are inadequate for this purpose, while the non-well-founded sets that he invented for the purpose (or should we say "discovered"?) worked marvelously. These non-well-founded sets are beautifully described by their membership diagrams, which are certain graphs. These non-sets, in the traditional view, would have to be called classes; but they do not even fit the usual view of classes as collections of well-founded sets. What are they? Structures which are used to describe classifiers.

In this view, a proper class is simply a class whose structure is too rich to permit it to be abstracted away, leaving only an extensional shell. The question of its being "too big" is not relevant; size is only one aspect of structure, and not always the most important. Of course, it may be that certain classes *are* "too big" to be sets; but the assumption that excessive size is the only reason why the abstraction operation might be undefined appears to be completely unjustified.

Hallett [25] has written a book which explores the history of the "limitation of size hypothesis", according to which every "inconsistent set" is inconsistent because it is "too big", i.e., contains a copy of the ordinals. He attributes this definition of "too big" to Jourdain, and the hypothesis itself to Russell [25, p. 183], although precursors of the idea are to be found in Cantor.

Once this misconception has been identified, one can see that it has pervaded the traditional interpretation of certain mathematical results. For example, Cantor's diagonal method shows that if we are given a sequence of real numbers x_n, we can construct another real number which is different from all the x_n. Does that mean that the set of real numbers is *larger* than the set of integers? That is the usual interpretation placed on the result. But it might be interpreted to mean instead that the set of reals has a very rich structure, which does not permit it to be covered by the integers.[14] This point of view finds some support from the fact that in constructive mathematics, for all we know every real number is recursive—and in that case, the reals would be in one-one correspondence with a subset of the integers (their recursive indices). They still could not be placed in recursive one-one correspondence with the integers, however, so Cantor's theorem would not be violated. How do we know that the actual situation is not better described in these terms, than in terms of size?

The theory formulated in the body of this paper, without the apparently artificial restriction that the formula in the separation axiom must not mention the application relation, is inconsistent. The contradiction appears to show that if we do not limit the axioms, the application relation, even restricted to sets of a fixed rank, cannot be a set. It is thus an example of a proper class which is not a set because of a structural complexity other than its size. Since the main piece of evidence for the "limitation of size hypothesis" seem to be Russell's observation that all the contradic-

[14] See [23, p. 230].

tory classes are "too big", we now have a specific reason for reconsidering that hypothesis.

Classes may have a more complex structure than can be summed up by telling what size they are. If we abstract away everything but the membership structure, we get sets. Operations have a more complex structure than their input-output relation, which is usually called the "graph". If we abstract that structure away, we get the usual set-theoretic representation. These abstractions have served mathematics well, but they may not be ideal for the foundations of computer science, and they may have led to mistakes in the foundations of mathematics.

References

[1] P. Aczel, *Lectures on Non-Well-Founded Sets* (Center for the Study of Language and Information, Stanford, CA, 1988).

[2] H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics* (North-Holland, Amsterdam, 1981).

[3] M. Beeson, Proving programs and programming proofs, in. *Logic, Methodology, and Philosophy of Science VII*, Proceedings of the meeting in Salzburg, Austria, July, 1983 (North-Holland, Amsterdam, 1986).

[4] M. Beeson, *Foundations of Constructive Mathematics: Metamathematical Studies* (Springer, Berlin, 1985).

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

[6] N.G. de Pruijn, The mathematical language AUTOMATH, its usage and some of its extensions, in: *Proc. Symp. on Automatic Demonstration*, IRAI, Versailles 1968, Lecture Notes in Mathematics **125** (Springer, Berlin, 1970) 29-61.

[7] L.E.J. Brouwer, Begründung der Mengenlehre unabhängig vom logischen Satz vom ausgeschlossenen Dritten. Erster Teil: Allgemeine Mengenlehre, Original date 1918, in: A. Heyting, ed., *L.E.J. Brouwer, Collected Works, Vol. I* (North-Holland, Amsterdam, 1975).

[8] R. Burstall and B. Lampson, A kernel language for modules and abstract data types, Tech. Rept., DEC Systems Research Center, Palo Alto, CA 94301, 1984.

[9] R. Constable et al., *Implementing Mathematics with the NuPrl Proof Development System* (Prentice-Hall, Englewood Cliffs, NJ, 1986).

[10] R. Constable, S. Johnson and C. Eichenlamb, *Introduction to the PL/CV2 Programming Logic*, Lecture Notes in Computer Science **135** (Springer, New York, 1982).

[11] R. Constable and D. Zlatin, The type theory of PL/CV3, in: *Proc. IBM Logic of Programs Conf.*, Lecture Notes in Computer Science **131** (Springer, Berlin, 1982) 72-93.

[12] H.B. Curry and R. Feys, *Combinatory Logic* (North-Holland, Amsterdam, 1958).

[13] R. Dedekind, *Essays on the Theory of Numbers* (Dover, New York, 1963) (Translations of: *Stetigkeit und irrationale Zahlen*, and: *Was sind und was sollen die Zahlen*).

[14] S. Feferman, A language and axioms for explicit mathematics, in: J. Crossley, ed., *Algebra and Logic*, Lecture Notes in Mathematics **450** (Springer, Berlin, 1975) 87-139.

[15] S. Feferman, Constructive theories of functions and classes, in: M. Boffa, D. van Dalen and K. McAloon, eds., *Logic Colloquium '78: Proc. Logic Coll. at Mons, 1978* (North-Holland, Amsterdam, 1979) 159-224.

[16] A. Fraenkel and Y. Bar-Hillel, *Foundations of Set Theory* (North-Holland, Amsterdam, 1958).

[17] G. Frege, *Grundgesetze der Arithmetik, begriffsschriftlich abgeleitet* (Verlag Hermann Pohle, Jena, 1893 (Band I), 1903 (Band II)), partial English translation in: P. Geach and M. Black, *Translations from the Philosophical Writings of Gottlieb Frege* (Blackwell, Oxford, 1980).

[18] H. Friedman, Some applications of Kleene's methods for intuitionistic systems, in: A. Mathias and H. Rogers, eds., *Cambridge Summer School in Mathematical Logic*, Lecture Notes in Mathematics **337** (Springer, Berlin, 1973) 113-170.

[19] H. Friedman, The consistency of classical set theory relative to a set theory with intuitionistic logic, *J. Symbolic Logic* **38** (1973) 315-319.
[20] H. Friedman, Set theoretic foundations for constructive analysis, *Ann. of Math.* **105** (1977) 1-28.
[21] H. Friedman and A. Ščedrov, The lack of definable witnesses and provably recursive functions in intuitionistic set theories, *Adv. in Math.* **57** (1985) 1-13.
[22] H. Graves, The Algos system, Tech. Rept., Language of Data Project, Los Altos, CA, 1985.
[23] N. Greenleaf, Liberal constructive set theory, in: *Proc. Constructive Mathematics*, New Mexico, 1980, Lecture Notes in Mathematics **873** (Springer, Berlin, 1981) 213-240.
[24] M. Gordon, R. Milner and C. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Lecture Notes in Computer Science **78** (Springer, Berlin, 1979).
[25] M. Hallett, *Cantorian Set Theory and Limitation of Size* (Clarendon Press, Oxford, 1984).
[26] J. van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1878-1931* (Cambridge University Press, Cambridge, MA, 1967).
[27] S. Hayashi, Extracting Lisp programs from constructive proofs: a formal theory of constructive mathematics based on Lisp, *Publ. Res. Inst. Math. Sci.* **19** (1983) 161-191.
[28] D. Hilbert, On the foundations of logic and arithmetic, in: [26] (original date 1904) 129-138.
[29] D. Hilbert, On the infinite, in: [26] (original date 1925) 367-392.
[30] S.C. Kleene, Recursive functionals and quantifiers of finite types I, *Trans. Amer. Math. Soc.* **91** (1959) 1-52.
[31] A.N. Kolmogorov, O principe tertium non datur (On the principle of tertium non datur), *Mat. Sb.* **32** (1925) 646-667 (Russian); English translation in: [26] 414-437.
[32] D.E. Knuth, Algorithmic thinking and mathematical thinking, *Amer. Math. Monthly* **92** (1985) 170-182.
[33] P. Martin-Löf, An intuitionistic theory of types: predicative part, in: H.E. Rose and J.C. Shepherdson, *Logic Colloquium '73* (North-Holland, Amsterdam, 1975) 73-75.
[34] P. Martin-Löf, Constructive mathematics and computer programming, in: L.J. Cohen, J. Los, H. Pfeiffer and K.P. Podewski, eds., *Logic, Methodology, and Philosophy of Science VI* (North-Holland, Amsterdam, 1982) 153-175.
[35] D. Mirimanoff, Les antinomies de Russell et de Burali-Forti et le problème fondamental de la théories des ensembles, *Enseign. Math.* **19** (1917) 37-52.
[36] A.F. Monna, The concept of function in the 19th and 20th centuries, in particular with regard to the discussions between Baire, Borel, and Lebesgue, *Arch. Hist. Exact Sci.* **9** (1972) 57-84.
[37] Y. Moschovakis, Axioms for computation theories—first draft, in: R. Gandy and M. Yates, eds., *Logic Colloquium '69* (North-Holland, Amsterdam, 1971) 199-256.
[38] L. von Neumann, An axiomatization of set theory, in: [26] pp. 393-413 (English translation of German original).
[39] G. Peano, The principles of arithmetic, presented by a new method, in: [26] (original date 1899) 83-97.
[40] E. Zermelo, Untersuchungen über die Grundlagen der Mengenlehre, *Math. Ann.* **65** (1908) 261-281: English translation: Investigations in the foundations of set theory I, in: [26], 199-215.
[41] J. Myhill, Constructive set theory, *J. Symbolic Logic* **40** (1975) 347-382.
[42] J. Smith, An interpretation of Marti-Löf's type theory in a type-free theory of propositions, *J. Symbolic Logic* **49** (1984) 730-753.
[43] K. Gödel, *The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis with the Axioms of Set Theory*, Annals of Mathematical Studies **3** (Princeton Univ. Press, 1940).