Relations Versus Functions at the Foundations of Logic: Type-Theoretic Considerations*

Paul E. Oppenheimer
Center for the Study of Language and Information
Stanford University
paul@plato.stanford.edu
and
Edward N. Zalta
Center for the Study of Language and Information
Stanford University
zalta@stanford.edu

Abstract

Though Frege was interested primarily in reducing mathematics to logic, he succeeded in reducing an important part of logic to mathematics by defining relations in terms of functions. By contrast, Whitehead & Russell reduced an important part of mathematics to logic by defining functions in terms of relations (using the definite description operator). We argue that there is a reason to prefer Whitehead & Russell's reduction of functions to relations over Frege's reduction of relations to functions. There is an interesting system having a logic that can be properly characterized in relational but not in functional type theory. This shows that relational type theory is more general than functional type theory. The simplification offered by Church in his functional type theory is an over-simplification: one can't assimilate predication to functional application.

*This paper is published in the *Journal of Logic and Computation*, 21 (2011): 351–374. At the end of the 3rd paragraph of Section 2.2, this preprint adds a left and right parenthesis, in red, to the logical type assigned to the Π function, thereby correcting an error. The authors would like to thank Uri Nodelman for his observations on the first draft of this paper. We'd also like to thank Bernard Linsky for observations on the second draft, which led us to reconceptualize the significance of our results within a more historical context. We'd also like to acknowledge one of the referees of this journal, whose comments led us to clarify and better document the claims in the paper.

1. Introduction

Although Frege failed to achieve his goal of reducing mathematics to logic, he did, however, "mathematize" an important part of logic.[1] Frege showed that one can reduce relations to functions and define natural language predication in terms of function application. For Frege, a simple natural language predication of the form 'John is happy' is analyzed as the application of the function denoted by '( ) is happy' to the object denoted by 'John'. The value of the function is a truth value. Similarly, a Fregean would analyze 'John loves Mary' in terms of a binary function that maps a pair of arguments to a truth value (or which maps its first argument to a unary function that maps its argument to a truth value). More recent analyses of these sentences, in which 'John' and 'Mary' are treated as generalized quantifiers (i.e., certain higher order functions), are variants of the Fregean analysis.

By contrast, Whitehead & Russell "logicized" an important part of mathematics by reducing mathematical functions to functional relations (see Linsky 2007). Those $n+1$-place relations R that obey the condition:

$$(Rx_1 \ldots x_n y \,\&\, Rx_1 \ldots x_n z) \rightarrow y = z$$

can represent n-ary functions. In classical logic, the atomic form of predication '$Rx_1 \ldots x_n$' is more fundamental than function application. Whitehead & Russell then use definite descriptions to define function application in terms of atomic predication. For them, 'John is happy' is analyzed in primitive notation as the predication '$H(j)$', and 'John loves Mary' is analyzed in primitive notation as the predication '$L(j,m)$', with no further analysis necessary or possible. (In what follows, we simplify this notation to 'Hj' and 'Ljm'.) Further, it is a consequence of the definition at PM, $*30 \cdot 01$ that we can define $f(x)$, when f is the functional relation R, as $\iota y(Rxy)$.[2] Under this analysis, the addition function becomes a 3-place functional relation, that is, $x+y$ is defined as $\iota z(Rxyz)$, where $+$ is analyzed as the 3-place functional relation R.

A key difference between the Fregean analysis and the Whitehead & Russell analysis is that, on the former, every formula becomes a term, that

[1] Hylton 1993 credits Burton Dreben for suggesting that the matter be put this way. See Hylton's footnote 28.

[2] In PM, $*30 \cdot 01$, we see that $R`y$ is defined as $\iota x(Rxy)$. But to get the definition in the text, we need to apply the converse operation at PM $*31 \cdot 02$ so that $f(x)$, when $f$ is a functional relation $R$, is defined as $\breve{R}`x$, i.e., $\iota y(Rxy)$.

is, a denoting expression. (For the remainder of this paper, we use the word 'term' to mean a denoting expression.) Whereas Frege thought that sentences named truth values, Whitehead & Russell, by contrast, don't require that formulas be terms or that they name anything. These two methods led to the development of two different kinds of type theories: relational type theories (RTTs) and functional type theories (FTTs). RTTs derive from Russell 1908, a variant of which serves as the basis of the type theory used in *Principia Mathematica*. FTTs derive from Church 1940, which clearly follows Frege in treating every formula as a term.

Given the Fregean reduction of relations to functions and the Russellian reduction of functions to relations, it is natural to think both that relations and functions are interdefinable, and that there is no significant difference between an RTT and an FTT. After all, doesn't the interdefinability show that for each RTT, there is an FTT which is a mere variant and vice versa, and that one can equally well start with relational types or functional types as the basis for the syntax and ontology of type theory? (We put aside here for the moment the complicating factor that Frege took functions to be extensional and Russell appeared to think of relations as intensional.) There are theorems about equivalences between RTTs and FTTs. For instance, Manzano (1996, 210–214) has proved the equivalence in expressive power of a particular FTT and a particular RTT. Andrews (2006), however, seems to suggest that the FTT based on Church 1940 is more general than the RTT derived from Russell 1908:

> In certain respects, it [Church's type theory] is simpler and more general than the type theory introduced by Bertrand Russell in Russell 1908 and Whitehead & Russell [1927]. Since properties and relations can be regarded as functions from entities to truth values, the concept of a function is taken as primitive in Church's type theory, ... (Andrews 2006, Introduction, paragraph 3)

In this paper we develop an argument that suggests Whitehead & Russell's approach to the foundations of logic, which takes relations as basic, is to be preferred to Frege's and Church's. Our argument goes by way of identifying an important difference between RTTs and FTTs with regard to their ability to represent systems containing formulas that aren't, and can't be converted to, terms. We argue that the logic of such systems, which is constituted by inferences between formulas, cannot be reduced to a logic constituted by inferences between terms.

In addition to using 'term' to refer to formal expressions that are assigned denotations in the semantics, we shall, in what follows, use 'formula' to refer to formal expressions that are assigned truth conditions.[3] In discussions and developments of FTTs, this distinction between formulas and terms is frequently not observed. See, for example, the locus classicus of simple FTT, namely, Church 1940. In what follows (Section 2.1), we define a simple RTT and develop its logic in the standard way: the fundamental axioms and rules of the logic apply to formulas. By contrast, in the standard formulation of FTT (Sections 2.2, 2.3), the fundamental axioms and rules of the logic apply to terms.

Before we present the main argument in Section 4.2, we prepare the reader (in Section 3) by describing and developing the properties of a particular system, namely, the object theory formulated in Zalta 1983, 1988 and elsewhere. The distinguishing feature of this system is the inclusion of formulas that are not terms and that cannot be converted to terms by λ-abstraction. A formula-based logic for this system is constructed in an RTT (Section 4.1). We then consider (Section 4.2) whether the logic of this system can be reformulated as an FTT and discover what appear to be insoluble problems in so doing. The problems became apparent to us when we attempted to extend the research program in computational metaphysics initiated in Fitelson & Zalta 2007. Our attempt to use an automated reasoning system based on FTT, namely, TPS (Andrews *et al*. 1966) ran into difficulties. There was no way to translate the terms and wffs of object theory into the language of TPS so as to capture all its inferences. This led us to suspect there was a deeper problem with the FTT underlying TPS. If we are correct, RTTs and FTTs are not, after all, mere variants; the conceptual framework embodied by RTT can represent logical inferences that FTT cannot properly represent.

In Section 5, we consider ways of undermining the argument of Section 4.2. We show that the suggested repairs to FTTs are not successful. Consequently, our efforts show an important difference between formula-based logics and term-based logics. This, in turn, will lead us to our

[3] In the theory of formal languages as developed in computer science, the word 'term' is used in a more general way. In this usage, a well-formed expression in a formal language is a term since it is a leaf of a formation tree. Thus the expressions we call formulas are terms in this sense. We are using the word 'term' in a narrower way than that. In our usage, terms are denoting expressions and formulas are expressions having truth conditions; not all well-formed formulas are terms.

conclusion (Section 6) about the significance of the Whitehead & Russell method of taking relations as basic in the foundations of logic.

2. Sketches of an RTT and an FTT

In this section, we sketch two simple (as opposed to ramified) type theories, one relational and one functional. Types will be used to categorize the domains of quantification as well as the classes of expressions that range over those domains.

2.1 A Sketch of a Representative RTT

We first sketch a relational theory RTT_0 of simple types. We define the types as follows:

- $\imath$ is a type.
- Where $\alpha_1, \ldots, \alpha_n$ are any types ($n \geq 0$), $\langle \alpha_1, \ldots, \alpha_n \rangle$ is a type.

Intuitively $\imath$ is the type for individuals and for the expressions that denote them. Also, intuitively, $\langle \alpha_1, \ldots, \alpha_n \rangle$ is the type for relations among objects of types $\alpha_1, \ldots, \alpha_n$, for any types $\alpha_1, \ldots, \alpha_n$ (and the type for expressions which denote such relations). Since we allow n to go to 0, $\langle\,\rangle$ is a type and we may introduce the type p by definition as follows:

- $p =_{df} \langle\,\rangle$

The type p is the type for propositions or truth-values, depending on whether one prefers an intensional or extensional interpretation of the system. Our claims in what follows hold on either interpretation. Notice that since the type p can be defined as above, no primitive type for propositions or truth values need be added to RTT.

Given this definition of types, we may assume that there are primitive constants and variables of every type. Then we may define 'formula' and 'term' simultaneously in the usual way:

1. (Simple) term: All primitive constants and variables of type α are (simple) terms of type α.
2. (Atomic) formula: If Δ is a term of type $\langle \alpha_1, \ldots, \alpha_n \rangle$, and $\tau_1, \ldots, \tau_n$ are terms of types $\alpha_1, \ldots, \alpha_n$, respectively, then $\Delta\tau_1 \ldots \tau_n$ is an (atomic) formula. Since we allow n to go to 0, when Δ is a term of type $\langle\,\rangle$ (i.e., type p), then Δ also is an (atomic) formula.
3. (Complex) formula: $\neg\phi$, $\phi \rightarrow \psi$, and $\forall\nu\phi$ are all (complex) formulas, whenever ϕ, ψ are formulas and ν is any variable of any type.
4. (Complex) term: Where ϕ is any formula and $\nu_1, \ldots, \nu_n$ ($n \geq 0$) are any variables (which may occur free in ϕ) of types $\alpha_1, \ldots, \alpha_n$, respectively, $[\lambda\nu_1 \ldots \nu_n\, \phi]$ is a (complex) term of type $\langle \alpha_1, \ldots, \alpha_n \rangle$.

It is absolutely essential to the points we make in what follows that terms of the form $[\lambda\nu_1 \ldots \nu_n\, \phi]$ be understood correctly. These terms are *not* function expressions. They are not to be confused with the expressions in the well-known "λ-calculus" (Church 1932, 1933, 1941; Kleene 1935). The above λ-expressions are to be read *relationally*. For example, to take a simple case, $[\lambda xy\, \neg Rxy]$ denotes the relation: being objects x and y that fail to exemplify the relation R. The claim $[\lambda xy\, \neg Rxy]ab$ asserts that objects a and b exemplify, or stand in, that relation. It does not assert anything about functions or function application! In general, $[\lambda\nu_1 \ldots \nu_n \phi]$ is to be read with a gerund: being objects $\nu_1, \ldots, \nu_n$ such that ϕ. Thus, $[\lambda xy \neg Rxy]ab$ is to be read: a and b exemplify the relation of being objects x and y that fail to exemplify the relation R. A predication of the form $[\lambda\nu_1 \ldots \nu_n\, \phi]\kappa_1 \ldots \kappa_n$ is to be read: $\kappa_1, \ldots, \kappa_n$ stand in the relation of being objects $\nu_1 \ldots \nu_n$ (respectively) such that ϕ. Of course, β-reduction holds for λ-expressions interpreted relationally. Where $\phi^{[\kappa_1,\ldots,\kappa_n]}_{[\nu_1,\ldots,\nu_n]}$ is the result of substituting the κ_i for the ν_i, respectively, in ϕ, β-reduction is captured as:

$$[\lambda\nu_1 \ldots \nu_n\, \phi]\kappa_1 \ldots \kappa_n \equiv \phi^{[\kappa_1,\ldots,\kappa_n]}_{[\nu_1,\ldots,\nu_n]}$$

To take our example from above as an instance, we read:

$$[\lambda xy\, \neg Rxy]ab \equiv \neg Rab$$

as: a and b exemplify the relation of being objects x and y that fail to exemplify the relation R if and only if a and b fail to exemplify the relation R. So far, this is all straightforward and well-known.

Note, however, that given clause (4) of the above definition, RTT_0 allows λ-expressions in which there are no variables bound by the λ. It is important to understand these properly. Just as the general type $\langle \alpha_1, \ldots, \alpha_n \rangle$ becomes the type $\langle\,\rangle$ (= type p) when $n = 0$, similarly, the

general form of λ-expressions $[\lambda\nu_1 \ldots \nu_n\, \phi]$ has as instances λ-expressions of the form $[\lambda\, \phi]$ when $n = 0$. Since $[\lambda\nu_1 \ldots \nu_n\, \phi]$ is a term of type $\langle\alpha_1, \ldots, \alpha_n\rangle$, $[\lambda\, \phi]$ is a term of type p. And as a term of type p, $[\lambda\, \phi]$ should be read: that-ϕ. For example, $[\lambda\, Po]$ might be read: that Obama exemplifies being President. To extend the example, if 'B' ('believes') is a 2-place relation of type $\langle\imath, p\rangle$, and 'j' ('John') is a term of type $\imath$, the formula $B(j, [\lambda Po])$ would be read: John believes that Obama is President.

It is important to note, however, that since they are terms of type p, expressions of the form $[\lambda\, \phi]$ also qualify as atomic formulas, by clause 2 of the above definition. That means that RTT has expressions (e.g., $[\lambda\, \phi]$) in the language which fall under two syntactic categories: formula and term. These expressions are typed insofar as they are terms, since only terms (i.e., denoting expressions) are typed. But by being both a term and a formula, one and the same expression should be read in two different ways, depending on whether it is being used as a term or as a formula. For example, in the previous paragraph, we saw that $[\lambda\, Po]$, when used as a term in the formula $B(j, [\lambda Po])$, was to be read 'that Obama exemplifies being President'. However, in the following formula, $[\lambda\, Po]$ is used as a formula:

$$[\lambda\, Po] \equiv Po$$

This is a well-defined complex sentence (indeed, as we shall see, it is an instance of β-reduction), in which the expression $[\lambda\, Po]$ on the left side of the biconditional should be read as a formula (i.e., it should express a thought). Here we must read $[\lambda\, Po]$ as: (the proposition) that Obama is President is true. This is perfectly coherent from a logical point of view, since the (non-semantic) concept of *truth* is the 0-place case of the concept of *exemplification*. In other words, the notion of exemplification expressed by the formula on the left-side of the biconditional:

$$[\lambda xy\, \neg Rxy]ab \equiv \neg Rab$$

becomes the concept of truth when expressed by formulas of the form:

$$[\lambda\, \neg Rab] \equiv \neg Rab$$

We've already seen that the former is read: a and b exemplify the relation of being objects x and y that fail to exemplify the relation R if and only if a and b fail to exemplify the relation R. But since exemplification

reduces to truth in the 0-place case, the latter is read: the proposition that a and b fail to exemplify the relation R is true if and only if a and b fail to exemplify the relation R. In general, then, the notion of exemplification expressed by formulas on the left-side of n-place β-reduction biconditionals of the form:

$$[\lambda\nu_1 \ldots \nu_n\, \phi]\kappa_1 \ldots \kappa_n \equiv \phi^{[\kappa_1,\ldots,\kappa_n]}_{[\nu_1,\ldots,\nu_n]}$$

becomes the concept of truth in the case of formulas on the left side of 0-place β-reduction biconditionals of the form:

$$[\lambda\, \phi] \equiv \phi$$

Indeed, we read $[\lambda\, \phi]$ as "the proposition that ϕ is true" anywhere it appears as a formula rather than in term position. As we shall see, these facts stand in contrast with FTT, where there is only one syntactic category ('term') rather than two and formulas are definable as a subset of the category 'term'.

One might wonder at this point whether one can eliminate the syntactic category 'formula' from RTT, say by assigning types to the connectives and quantifiers. This isn't usually done because syntactically the connectives and quantifiers are functions from formulas to formulas and RTT is not the natural environment for assigning these expressions denotations that are *functionally* typed. Moreover, the literature shows that this is not the usual way to formulate RTT. One difference between RTT and FTT in Manzano 1996 is due to the fact that the connectives and quantifiers of RTT are not assigned types.[4] Muskens 1989 doesn't include primitive formulas with connectives and quantifiers in his language for RTT.[5] Although we haven't formulated RTT so that the connectives and quantifiers become terms assigned particular types, our argument in Section 4.2 is designed to establish that one should not do so, since RTT is more flexible in the absence of such typing. Indeed, one might reasonably

[4] See rules (E5) and (E6) on pp. 188-189.

[5] See Definition 10, p. 12. In Definition 16 and 17, formulas involving connectives and quantifiers become defined. Note that Muskens' relational type theory is relational only because the types are interpreted relationally; no primitive atomic formula of his language expresses a statement to the effect that objects are related by a relation R (the primitive identity formulas defined in clause iii in Definition 10 are not parsed as relation statements in which '=' is a 2-place relation). In what follows, we shall assume that a true relational type theory not only has relational types but also has relational atomic formulas, that is, formulas of the kind that can express the thought that some objects exemplify or stand in relation R.

stipulate that one of the principal distinctions between RTT and FTT is that RTT includes the primitive syntactic category 'formula' and does not type the connectives and quantifiers. We hope to show that it is a mistake to eliminate this primitive syntactic category in favor of the single primitive category 'term', as is done in FTT.

We shall not dwell here on any more specifics of RTT_0. For example, we need not consider other kinds of complex terms in addition to λ-expressions, such as definite descriptions, since they are straightforward. Given the above well-defined notions of term and formula, the logic of RTT_0 can be developed as a logic of formulas, that is, as a set of formulas designated as axioms and inference rules that are relations among formulas. Thus we shall assume:

1. The classical axioms and rules for propositional logic, including all tautologies as axioms and the rule of Modus Ponens

2. The classical axioms and rules for predicate logic, including the rule of Generalization

3. The classical axioms governing formulas containing λ-expressions, including β-reduction and α- and η-conversion.

In the usual way, one defines a proof in RTT_0 as a sequence of formulas such that each member of the sequence either is an axiom or follows from previous members of the sequence by a rule of inference.

2.2 A Sketch of a Representative FTT

Now let's sketch a simple functional type theory FTT_0. We basically follow Church 1940, though we use the recursive style which is now standard. One need not follow Church to make our point; what is essential is that our representative FTT be developed in such a way that the formulas of the language form a subclass of the terms, i.e., they are defined as terms that denote a truth-value.

We define the following types:[6]

[6]This definition of types differs from that of Church. Church requires all complex types to be of the form $(\alpha\beta)$, and "functions of several variables are explained, after Schönfinkel [1924], as functions of one variable whose values are functions" (Church 1940, 57). To facilitate the comparison with RTT, we have defined 'multi-argument' functional types to characterize functions of several variables.

- $\imath, o$ are both primitive types.

- If $\alpha_1, \ldots, \alpha_n, \beta$ are any types, then $(\beta, \alpha_1, \ldots, \alpha_n)$ is a complex type $(n \geq 1)$.

Intuitively, $\imath$ is the type for individuals, and o is the type for truth-values or propositions (depending on whether the system is construed extensionally or intensionally). Similarly, intuitively, $(\beta, \alpha_1, \ldots, \alpha_n)$ is the type of n-ary functions which take as arguments a sequence (of length n) of objects of types $\alpha_1, \ldots, \alpha_n$, respectively, and which have as values objects of type β.

The symbols of the language include: improper symbols (,), [,], and λ; and the proper symbols, which include (a) variables of every type, and (b) the logical constants: $\neg$ (of type (o, o)), $\rightarrow$ (of type (o, o, o)), and Π (of type $(o, (o, \alpha_1, \ldots, \alpha_n))$).

We define terms as follows:

1. Simple terms of type α: All primitive constants and variables of type α are terms of type α.

2. Atomic function application: Where $\tau_1, \ldots, \tau_n$ are variables of types $\alpha_1, \ldots, \alpha_n$ and Δ is a term of type $(\beta, \alpha_1, \ldots, \alpha_n)$, then $\Delta(\tau_1, \ldots, \tau_n)$ is a term of type β.

3. Complex terms of type $(\beta, \alpha_1, \ldots, \alpha_n)$: Where ϕ is any term of type β and $\nu_1, \ldots, \nu_n$ are any variables of types $\alpha_1, \ldots, \alpha_n$, respectively, (which may or may not occur free in ϕ) then $[\lambda\nu_1 \ldots \nu_n\ \phi]$ is a (complex) term of type $(\beta, \alpha_1, \ldots, \alpha_n)$.

There are four observations to make about this definition.

1. Terms (including terms of type o) can be open (i.e., have free occurrences of variables) or closed.

2. One can pick out a class of terms of FTT that correspond to the formulas of RTT, namely, terms of type o, i.e., terms that denote truth-values (relative to an assignment to the variables). We may call any type-o term of the form $f(\tau_1 \ldots \tau_n)$ an 'atomic formula' whenever f is a term of type $(o, \alpha_1, \ldots, \alpha_n)$ and $\tau_1, \ldots, \tau_n$ are terms of types $\alpha_1, \ldots, \alpha_n$, respectively.

3. The expressions of FTT which correspond to the 'molecular formulas' of RTT are terms. Negation ($\neg$) and conditionalization ($\rightarrow$) are simply represented as functions of type (o, o) and type (o, o, o), respectively. So where ϕ is a term of type o, $\neg\phi$ is also a term of type o. Where, ϕ, ψ are terms of type o, so is $\rightarrow \phi\, \psi$. (In what follows, we shall put $\rightarrow$ in infix notation, abbreviating $\rightarrow \phi\, \psi$ as $\phi \rightarrow \psi$.)

4. The expressions of FTT which correspond to the quantified formulas of RTT also become terms. We may define the variable-binding universal quantifier '$\forall$' by letting

   $$\forall \nu_1 \ldots \nu_n \phi$$

   (where ϕ is a term of type o and $\nu_1, \ldots, \nu_n$ are terms of types $\alpha_1, \ldots, \alpha_n$) abbreviate the following term of type o

   $$\Pi([\lambda \nu_1 \ldots \nu_n\ \phi]).$$

   Intuitively, the formula $\Pi([\lambda \nu_1 \ldots \nu_n\, \phi])$ asserts that every sequence of elements of types $\alpha_1, \ldots, \alpha_n$ satisfies ϕ.

Note that Church (1940) didn't take 'formula' (in our sense, as an expression that is assigned satisfaction, or truth, conditions) to be a primitive syntactic type in his system. Instead, every denoting expression in his system is a 'well-formed formula'. We will not adopt this use of 'formula'. In what follows, the terms of FTT_0 are just its denoting expressions, and the well-formed formulas of FTT_0 are restricted to terms of type o (we may call the other well-formed expressions 'well-formed terms').

2.3 A Term Logic For FTT_0

To develop a term logic for FTT_0,[7] we use the standard definitions for the truth functional connectives. The rules of inference are:

1. **Modus Ponens**. From ϕ and $\phi \rightarrow \psi$ infer ψ.

2. **Generalization**. From ϕ infer $\forall \nu \phi$, where ν is of any type.

The axiom schemata are:

1. All tautologies are axioms.

[7] See Church 1940, 60–63; Andrews 2002, 204.

2. $\forall \nu \phi \rightarrow \phi[\tau/\nu]$, where τ is any term substitutable for ν.

3. $\forall \nu(\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \forall \nu \psi)$, provided ν is not free in ϕ.

4. $[\lambda \nu_1 \ldots \nu_n\ \phi](\mu_1, \ldots, \mu_n) \equiv \phi'$, where ϕ is a term of type o, and $\nu_1, \ldots, \nu_n$ and $\mu_1, \ldots, \mu_n$ are variables of types $\alpha_1, \ldots, \alpha_n$, and ϕ' is the result of substituting $\mu_1, \ldots, \mu_n$ simultaneously for $\nu_1, \ldots, \nu_n$ in ϕ.

5. $[\lambda \nu_1 \ldots \nu_n\, \Delta(\nu_1, \ldots, \nu_n)] = \Delta$, where $\nu_1, \ldots, \nu_n$ are any variables of types $\alpha_1, \ldots, \alpha_n$, and Δ any function term of type $(\beta, \alpha_1, \ldots, \alpha_n)$, and $[\lambda \nu_1 \ldots \nu_n\, \Delta(\nu_1, \ldots, \nu_n)]$ is an 'elementary' λ-expression (i.e., one where the matrix is an atomic formula).

6. $[\lambda \nu_1 \ldots \nu_n\ \phi] = [\lambda \mu_1 \ldots \mu_n\ \phi']$, where $\mu_1, \ldots, \mu_n$ are variables that are substitutable for the variables $\nu_1, \ldots, \nu_n$, respectively, in ϕ and ϕ and ϕ' are alphabetic variants with respect to the ν_i and μ_i.

Axioms 4, 5, and 6 ground the rules of β and η reduction, and α conversion, respectively. Note that we have not asserted any axiom of extensionality, so as to leave open the possibility of an intensional interpretation of the formalism. Note also that although the definition of a proof is identical to that for RTT_0, the distinguishing feature of FTT_0's logic is that every line of every proof is a term, and so the rules of inference are relations between terms. In RTT_0, formulas are not terms, so the inferential relations between formulas are not inferential relations between terms. The important point here is that the inferences of FTT_0 are between syntactic expressions which must be terms *as well as* formulas. As we shall see, this is a crucial difference between RTT_0 and FTT_0.

2.4 Important Differences Between RTTs and FTTs

It is natural to observe that:

(a) There is a subsystem of each FTT which is an RTT, namely, the subsystem that results when that FTT is restricted to the functional types of the form $(\beta, \alpha_1, \ldots, \alpha_n)$ when $\beta = o$, and

(b) There is a subsystem of each RTT which is an FTT, namely, the subsystem that results when that RTT is restricted to functional

relations.[8]

We think many readers will conclude from this observation that it doesn't matter whether we formulate type theory relationally or functionally and thus that it doesn't matter whether we start with relations or functions at the foundation of logic. We also suspect that many readers (e.g., those trained in computer science departments) may prefer the 'Fregean' method of mathematizing logic by taking mathematical functions as basic. We hope to show that this conclusion and preference are mistaken by providing evidence for thinking that the use of relations as primitive is logically more fundamental and offers a better foundation for logic and mathematics than the use of functions as primitive. We think it is important not to collapse the distinction between terms and formulas because that collapses the distinction between naming and predication, resulting in the undesirable consequences described below.

The following observations, about how RTT_0 and FTT_0 reveal distinctions between RTTs and FTTs, will be salient for our argument in Section 4.2. These observations concern the differences between relations and functions.

One crucial difference between RTTs and FTTs is that the former have no need of a special primitive type for propositions, whereas FTTs do need a distinguished primitive type for truth-values (or propositions), In RTT_0, propositions come for free as 0-place relations, i.e., relations of type p.[9] Moreover, in FTT_0 it would be impossible to formulate statements about anything without introducing a primitive type for truth-values or propositions.

Another crucial difference between RTTs and FTTs is that in the former, it is possible to assert formulas without those formulas being

[8] A functional relation, in the context of an RTT, is any relation R of type $\langle \alpha_1, \ldots, \alpha_n, \alpha_{n+1} \rangle$ (for $n \geq 1$) such that, for any objects $x_1, \ldots, x_n, y, z$, where $x_1, \ldots, x_n$ are of types $\alpha_1, \ldots, \alpha_n$, respectively, and y, z are of type α_{n+1}, R satisfies the condition: $Rx_1 \ldots x_n y \,\&\, Rx_1 \ldots x_n z \rightarrow y = z$.

[9] In FTTs, the empty sequence of types can't be a type. In the clause in the definition of types that asserts:

> If $\alpha_1, \ldots, \alpha_n, \beta$ are any types, then $(\beta, \alpha_1, \ldots, \alpha_n)$ is a complex type $(n \geq 1)$.

we can't let n go to 0. Though logicians and computer scientists sometimes *extend* the notion of function so that constants are treated as 0-place functions, such an extension is not grounded in Frege's idea of a function as a map from arguments to values. Given that idea of a function, if there are no arguments to be mapped, then there is no function.

names of anything; i.e., a formula must have truth conditions but need not have a denotation. The definition of truth yields truth conditions for every formula ϕ, but the definition of denotation (for terms) need not assign formulas denotations. By contrast, in FTT_0, every formula must be assigned a denotation.[10]

It is also worth observing that relations can be understood primarily as intensional entities that *characterize* their relata. By contrast, mathematical functions are typically understood as extensional entities that simply *correlate* their arguments and values.[11] This difference between our intuitive notions of characterization and correlation is philosophically important, for the former is tied to the notion of predication while the latter is tied to the notion of functional application. In a predication, the properties and relations denoted by the predicates characterize the objects denoted by the subject and object terms. But in the corresponding analysis of statements in terms of functional application, the function does nothing more than map its arguments to propositions or truth-values. For example, the primitive predications of the form Fa and Rab in RTTs say, respectively, that the property F characterizes the object a and that the relation R characterizes the related objects a and b. No such intuitive reading is assumed for the FTT formulas Fa and Rab.

Of course, this 'intended interpretation' of RTT may not carry much weight with some logicians and computer scientists. They may note that for all serious model-theoretic investigations, the truth conditions of the

[10] This difference may hold the key to the question of whether Russell's own type theory was a relational type theory, given that he seemed more inclined to work with propositional functions than with relations. This question of Russell scholarship goes beyond the scope of this paper. One might argue that Russell developed an RTT from the fact that this difference, and the one just previously mentioned, apply to his type theory. Russell can be read as holding the view that there is no special distinguished type for propositions that serves as the type of outputs of propositional functions, in which case, propositional functions are nothing other than relations, with propositions as 0-place relations. Moreover, the formulas of PM aren't (convertible into) terms that denote propositions.

[11] Church (1940) seems to describe his theory of simple types as open to interpretations in which the types can be either extensional or intensional. If the logical functions (predication, the truth functions, and quantification) are defined as particular mathematical functions, then even if the outputs of the logical functions are intensional objects, those functions are still mere correlations, and hence extensional. We'll consider below whether the use of intensional propositions instead of extensional truth-values as a basic type in FTT_0 offers a solution to the problem we develop. (It doesn't.)

primitive relational statement Rxy is given by the claim that the ordered pair consisting of the objects assigned to x and y (in that order) is an element of the set of ordered pairs assigned to R. And then it will be noted that R can be assigned a function instead, namely, the characteristic function of the set of ordered pairs assigned to R. But our point here is only that the following facts do *not* imply that relations *just are* functions or that functions *just are* relations:

- that relations and functions are interdefinable (when considered extensionally),
- that RTT_0 has a subsystem that constitutes an FTT, and
- that FTT_0 has a subsystem that constitutes an RTT.

From the point of view of philosophical logic, one may conceive functions and relations as very different fundamental entities and one *may* therefore interpret RTTs very differently from FTTs. There are real philosophical differences between taking relations as primitive and taking functions as primitive. We plan to show, in what follows, that there is reason to think that these two approaches are not equivalent as foundations for logic.

It is important to mention one final group of interesting features concerning RTTs and FTTs. RTTs have two basic syntactic categories (formula and term), and may include expressions that are categorized as both formulas and terms. We saw examples of this earlier (Section 2.1), when we discussed the fact that the expression $[\lambda\, \phi]$ (in which the λ binds no variables) is both a term (i.e., denoting expression) and a formula (expression assigned truth conditions). So in RTTs, expressions can be of more than one syntactic category (though as terms, they can have only one type). By contrast, FTTs don't have 'formula' as a primitive syntactic category. Formulas are defined as special kinds of terms, and so FTT doesn't include expressions that fall under more than one syntactic category.

3. A Theory of Abstract Objects[12]

3.1 The language of the theory

So as to make the present paper self-contained, we review the most important elements of the theory of abstract objects (Zalta 1983, 1988). We shall henceforth call this theory 'object theory'.[13] In its simplest form, this theory is couched in a second-order modal language (without identity) modified only so as to admit a second kind of atomic formula. This distinctive feature, namely, the presence of two forms of atomic predication, derives from Mally (1912). This distinction between two kinds of predication is preserved in the formulation of higher-order object theory, but for now, let's focus on the simpler second-order version of the theory.

To introduce these two kinds of atomic formulas, let $\kappa_1, \kappa_2, \ldots$ be any object terms (either primitive object terms such as constants $a, b, c, \ldots$ or variables $x, y, z, \ldots$, or complex object terms such as definite descriptions, to be defined below) and let $\Theta^n_1, \Theta^n_2, \ldots$ be any n-place relation terms, $n \geq 0$ (either primitive relation terms, such as constants $P^n, Q^n, \ldots$ or variables $F^n, G^n, \ldots$, or complex relation terms, such as λ-expressions, to be defined below). Then the language of object theory will include both the usual atomic formulas of the form $\Theta^n\kappa_1 \ldots \kappa_n$ and new atomic formulas of the form $\kappa\Theta^1$.[14] In the definition of a second-order language with encoding, there are two base clauses for atomic formulas:

> Where Θ^n is any n-place relation term and $\tau_1, \ldots, \tau_n$ are any object terms, then $\Theta^n\tau_1 \ldots \tau_n$ is an atomic (exemplification) formula.
>
> Where Θ^1 is any 1-place relation term and τ is any object term, then $\tau\Theta^1$ is an atomic (encoding) formula.

We read the first kind of atomic formula as: objects $\kappa_1, \ldots, \kappa_n$ *exemplify* the relation Θ^n. We read the second kind of atomic formula as: object κ *encodes* property Θ^1. We will discuss the new atomic encoding formulas in more detail below. But given these atomic formulas as the base case of a recursive definition, the molecular, quantified and modal formulas are

[12] This section has been added at the request of the referees.

[13] The objects of this theory are not to be confused with the objects of object-oriented programming languages, nor with the objects that are instances of abstract data types.

[14] In the case where $n = 0$, the first kind of formula, $\Theta^n\kappa_1 \ldots \kappa_n$, becomes the 0-place relation (= propositional) term Θ^0, which may be, for example, either the propositional constant P^0 or the propositional variable F^0.

the usual ones: if ϕ and ψ are any formulas and ν any object or relation variable, then $\neg\phi$, $\phi \rightarrow \psi$, $\forall\nu\phi$, and $\Box\phi$ are formulas. In what follows, we suppose $\phi \,\&\, \psi$, $\phi \vee \psi$, and $\phi \equiv \psi$ are defined in the usual way.

In object theory the notions of 'formula' and 'term' are defined simultaneously, and the definition of 'term' has clauses for the formation of definite descriptions and λ-expressions. We therefore use the following two clauses to introduce the complex terms:

1. Where ν is any object variable and ϕ is any formula, then $\imath\nu\phi$ is a complex object term. We read $\imath\nu\phi$ as: the object ν such that ϕ.

2. Where $\nu_1, \ldots, \nu_n$ are any object variables and ϕ is any formula that has no encoding subformulas, then $[\lambda\nu_1, \ldots, \nu_n\ \phi]$ is a complex n-place relation term (for $n \geq 0$).

We read the expression $[\lambda\nu_1, \ldots, \nu_n\ \phi]$ in either of the following ways:

- being objects $\nu_1, \ldots, \nu_n$ such that ϕ
- the relation that objects $\nu_1, \ldots, \nu_n$ exemplify just in case ϕ

For example:

- The λ-expression $[\lambda x\ \neg Rx]$ might be read: being an object x such that x fails to exemplify being round
- The λ-expression $[\lambda xy\ Rx \,\&\, \neg Py]$ might be read: being objects x and y such that x exemplifies R and y fails to exemplify P.
- The λ-expression $[\lambda xy\ Rx \,\&\, \neg Py]$ might be read: being objects x and y such that x exemplifies R and y fails to exemplify P.

Clearly, we are understanding λ-expressions *relationally* and not functionally, i.e., as denoting relations rather than functions.

In the clause introducing λ-expressions, we let 'subformula' have the simple definition:

1. If ϕ is any of the formulas, $\neg\psi$, $\psi \rightarrow \chi$, $\forall\nu\psi$, or $\Box\psi$, then ψ (χ) is a subformula of ϕ.

2. If ψ is a subformula of χ and χ is a subformula of ϕ, then ψ is a subformula of ϕ.

Notice that the encoding formula 'xG' is not a subformula of '$F\imath x(xG)$', since it is embedded in a term within the formula. Thus λ-expressions such as $[\lambda y\ Ry\imath x(xG)]$ are well-formed. The matrix $Ry\imath x(xG)$ has no encoding subformulas, given our definition. However, the 'no encoding subformulas' restriction implies that the expression $[\lambda xy\ Px \rightarrow \Box yQ]$ is not well-formed: 'yQ' is a subformula of '$\Box yQ$' formula, and '$\Box yQ$' is a subformula of '$Px \rightarrow \Box yQ$', making 'yQ' a subformula of '$Px \rightarrow \Box yQ$'. Since this latter has an encoding subformula, it is not allowed in any λ-expressions of the form $[\lambda\nu_1, \ldots, \nu_n\ \phi]$ ($n \geq 0$).

The simultaneous definition of formula and term yields following examples of formulas:

- $[\lambda y\ Ray \,\&\, \neg\exists z(Qzy)]b$. This is an atomic exemplification formula with a complex relation term. It asserts that object b exemplifies the (complex) property of being an object y such that a bears R to y and such that it is not the case that something bears Q to y.
- $\imath x(Rx \,\&\, Qx)P$. This is an atomic encoding formula with a complex object term. It asserts that the object x which is both R and Q encodes the property P.
- $\forall F(aF \rightarrow Fa)$. This quantified formula asserts that object a exemplifies every property that it encodes.
- $\Box\exists x(A!x \,\&\, \forall F(xF \equiv Fb))$. This modal, quantified formula asserts that necessarily, there is an object x that exemplifies the property $A!$ and which is such that it encodes all and only the properties exemplified by object b.

Before we turn to some of the important definitions in object theory, such as that of identity, it would serve well to examine the effect of letting n go to 0 in $[\lambda\nu_1, \ldots, \nu_n\ \phi]$. $[\lambda\ \phi]$ is an acceptable λ-expression and it denotes a 0-place relation (i.e., a proposition). Recalling our work in Section 2.1, this λ-expression is to be read as: that ϕ. For example, the permissible expression $[\lambda\ Pa]$, where λ binds no variables is read: that a exemplifies P. For purposes of clarity, we stipulate that the λ takes precedence over all the connectives and quantifiers in the matrix governed by the λ. Thus, in the expression $[\lambda\ Pa \rightarrow \Box Qb]$, the entire formula following the λ falls within the scope of the λ and so the expression denotes the conditional proposition that if a exemplifies P then necessarily

b exemplifies Q. Note, finally, that the 0-place λ-expressions like $[\lambda Pa]$ are to be distinguished from 1-place λ-expressions like $[\lambda x\ Pa]$. The former denotes a proposition, whereas the latter denotes a property. The former denotes the proposition that a exemplifies P, whereas the latter denotes the property of being an object x such a exemplifies P. The difference here is crucial, since the former expression denotes a 0-place relation while the latter denotes a 1-place relation.

In the next series of definitions, we assume that there is a single distinguished one-place predicate, namely, $E!$. For present purposes, we may interpret $E!x$ as asserting that x exemplifies the property of being concrete. We may therefore define two kinds of objects: the ordinary objects are those that might possibly be concrete (i.e., $O!x =_{df} \Diamond E!x$) and the abstract objects are those that couldn't possibly be concrete ($A!x =_{df} \neg\Diamond E!x$). This distinction is crucial to the theory, for whereas ordinary objects only exemplify properties, abstract objects may encode as well as exemplify properties. There will be a comprehension principle that asserts that for any condition on properties ϕ, there is an abstract object that encodes exactly the properties that satisfy (in Tarski's sense) the condition ϕ.

We may now define identity for objects, properties, relations, and propositions, respectively:

- $x = y =_{df}$
  $[O!x \,\&\, O!y \,\&\, \Box\forall F(Fx \equiv Fy)] \vee [A!x \,\&\, A!y \,\&\, \Box\forall F(xF \equiv yF)]$
- $F^1 = G^1 =_{df} \Box\forall x(xF \equiv xG)$
- $F^n = G^n =_{df}$ [omitted for simplicity]
- $F^0 = G^0 =_{df} [\lambda x\ F^0] = [\lambda x\ G^0]$

The first definition tells us that objects x and y are identical iff either (a) they are both ordinary objects that necessarily exemplify the same properties, or (b) they are both abstract objects that necessarily encode the same properties. The second definition asserts that properties F^1 and G^1 are identical whenever they are necessarily encoded by the same objects. The third definition is omitted for simplicity, but basically, F^n and G^n are identical whenever each way of plugging $n-1$ objects into F^n and G^n (plugging them in the same order) always results in two 1-place

properties that are identical according to the definition of $F^1 = G^1$.[15] Finally, the fourth definition asserts that propositions F^0 and G^0 are identical just in case the property *being an x such that* F^0 is identical to the property of *being an x such that* G^0, thereby reducing it to a previous case.

3.2 The Semantics

We interpret the language of object theory in the most straightforward way possible, namely, by assuming that the object variables and constants range over a domain of objects and that the relation variables and constants range over a domain of relations among those objects. To be more precise, let the object terms take denotations in, or range over, the members of a domain of objects $\mathbf{D}$, and the n-place relation terms take denotations in, or range over, the members of a domain of n-place relations $\mathbf{R_n}$ (and let $\mathbf{R} = \bigcup_{\mathbf{n}\geq 0} \mathbf{R_n}$). Assuming a primitive domain of possible worlds, each n-place relation is assigned an exemplification extension that can vary from world to world, while each 1-place property is assigned an encoding extension that doesn't vary with the worlds. Thus, a function $\mathbf{ext_w}$ assigns, at each world $\mathbf{w}$, a set of n-tuples (whose elements are in $\mathbf{D}$) to each $\mathbf{r^n}$ in $\mathbf{R_n}$. A second function, $\mathbf{ext_A}$ assigns a subset of $\mathbf{D}$ to each member $\mathbf{r^1}$ in $\mathbf{R_1}$. So this second function is independent of the possible worlds, and will be used to interpret atomic encoding formulas. Now, given (a) some interpretation function $\mathbf{I}$ that assigns each object and relation constant to a member of the appropriate domain, (b) some appropriate assignment $\mathbf{f}$ to the variables, and (c) a denotation function $\mathbf{d_{I,f}}$ which agrees with the interpretation function $\mathbf{I}$ on the constants and agrees with the assignment function $\mathbf{f}$ on the variables, we may interpret our atomic formulas by defining satisfaction with respect to a world, as follows (ignoring the presence of definite descriptions, for simplicity):

1. Where ϕ is an atomic exemplification formula of the form $\Theta^n\kappa_1 \ldots \kappa_n$, $\mathbf{f}$ satisfies ϕ with respect to world $\mathbf{w}$ iff $\langle \mathbf{d_{I,f}}(\kappa_1), \ldots, \mathbf{d_{I,f}}(\kappa_n)\rangle \in \mathbf{ext_w}(\mathbf{d_{I,f}}(\Theta^n))$

[15] See Zalta 1993, note 21. An example illustrates the definition. For F^3 and G^3 to be identical, the following identities must hold (for arbitrarily chosen objects x, y): (a) $[\lambda z\, Fzxy] = [\lambda z\, Gzxy]$, (b) $[\lambda z\, Fxzy] = [\lambda z\, Gxzy]$, and (c) $[\lambda z\, Fxyz] = [\lambda z\, Gxyz]$. This reduces relation identity to property identity.

2. Where ϕ is an atomic encoding formula of the form $\kappa\Theta^1$, $\mathbf{f}$ satisfies ϕ with respect to world $\mathbf{w}$ iff $\mathbf{d_{I,f}}(\kappa) \in \mathbf{ext_A}(\mathbf{d_{I,f}}(\Theta^1))$

These definitions can be modified in the usual way to allow for non-denoting definite descriptions and terms that might contain them.[16] Note that worlds $\mathbf{w}$ play no role in the definition of satisfaction for atomic encoding formulas, since the properties an object encodes will not vary from world to world.

3.3 Logic

The logic for this language is as simple as it could be modulo the presence of definite descriptions which are interpreted as rigidly designating the unique objects satisfying their matrices. No changes to the axioms and rules of second-order logic are required to accommodate atomic encoding formulas, though there is a special modal axiom for encoding formulas.

We take as axioms the modal closures of all the usual axioms of classical propositional logic, classical quantification theory, and classical S5 modal logic. To this basis, we add all of the instances of the following modal principle for encoding:

$$\Diamond xF \rightarrow \Box xF$$

In other words, encoding is not relative to any circumstance. This is validated semantically by the fact that the satisfaction conditions, at a world, for atomic encoding formulas are independent of the worlds.

Given that we have two sorts of complex terms in the language, we may conclude the statement of the axioms by saying which axioms govern the λ-expressions and the definite descriptions. λ-expressions are governed by the modal closures of the usual axioms that ground the derived rules for λ-conversion (β reduction, η reduction, and α conversion, respectively):

(β) $[\lambda x_1 \ldots x_n\ \phi]y_1 \ldots y_n \equiv \phi^{y_1,\ldots,y_n}_{x_1,\ldots,x_n}$ (ϕ free of descriptions)

[16] The definitions then become:

1. Where ϕ is an atomic exemplification formula of the form $\Theta^n\kappa_1 \ldots \kappa_n$, $\mathbf{f}$ satisfies ϕ with respect to world $\mathbf{w}$ iff $\exists \mathbf{o_1} \ldots \exists \mathbf{o_n} \in \mathbf{D}, \exists \mathbf{r_n} \in \mathbf{R}[\mathbf{o_1} = \mathbf{d_{I,f}}(\kappa_1)$ and $\ldots$ and $\mathbf{o_n} = \mathbf{d_{I,f}}(\kappa_n)$ and $\mathbf{r_n} = \mathbf{d_{I,f}}(\Theta^n)$ and $\langle \mathbf{o_1}, \ldots, \mathbf{o_n} \rangle \in \mathbf{ext_w}(\mathbf{r_n})]$.
2. Where ϕ is an atomic encoding formula of the form $\kappa\Theta^1$, $\mathbf{f}$ satisfies ϕ with respect to world $\mathbf{w}$ iff $\exists \mathbf{o} \in \mathbf{D}, \exists \mathbf{r^1} \in \mathbf{R}[\mathbf{o} = \mathbf{d_{I,f}}(\kappa)$ and $\mathbf{r^1} = \mathbf{d_{I,f}}(\Theta^1)$ and $\mathbf{o} \in \mathbf{ext_A}(\mathbf{r^1})]$.

(η) $[\lambda x_1 \ldots x_n\ F^n x_1 \ldots x_n] = F^n$

(α) $[\lambda x_1 \ldots x_n\ \phi] = [\lambda x'_1 \ldots x'_n\ \phi']$ (ϕ, ϕ' alphabetic variants in x, x')

Since the definite descriptions of the form $\imath x\phi$ are treated semantically as rigidly designating the object that satisfies the matrix at the actual world, they are governed by an axiom schema the instances of which are contingent, i.e., the instances of the following axiom schema governing descriptions are logical truths that are not necessary (Zalta 1988, 90):

> Descriptions: $\psi^{\imath x\phi}_y \equiv \exists x(\phi\ \&\ \forall z(\phi^z_x \rightarrow z = x)\ \&\ \psi^x_y)$, for any atomic formula or defined identity formula $\psi(y)$ in which y is free.

Consequently, we take as axioms only the instances of Descriptions, not arbitrary modal closures of those instances.

To these axioms, we annex the usual rules of MP and GEN, and complete the logic with a Rule of Necessitation (RN) restricted only so that it does not apply to any line that depends on an instance of the Descriptions axiom (since those instances are not necessary truths).

3.4 The Proper Theory and Its Consequences

There are two non-logical axioms in object theory:

1. $O!x \rightarrow \Box\neg\exists F\, xF$
2. $\exists x(A!x\ \&\ \forall F(xF \equiv \phi))$, where ϕ has no free xs

The first asserts that ordinary objects necessarily fail to encode properties. Encoding is a mode of predication that applies only to abstract objects. The second proper axiom is the comprehension schema for abstract objects. It asserts, for any condition ϕ, that there exists an abstract object that encodes all and only the properties satisfying ϕ. Here are some examples of comprehension, where 'p' in the following formulas ranges over propositions (0-place relations) and $\approx_E$ is the relation that two properties F and G bear to one another just in case there is a one-one correspondence between the ordinary objects exemplifying them:

- $\exists x(A!x\ \&\ \forall F(xF \equiv Fy))$
- $\exists x(A!x\ \&\ \forall F(xF \equiv \exists p(p\ \&\ F = [\lambda z\ p])))$
- $\exists x(A!x\ \&\ \forall F(xF \equiv \Box\forall y(Gy \rightarrow Fy)))$

- $\exists x(A!x \,\&\, \forall F(xF \equiv F \approx_E G))$

The first asserts the existence of an abstract object that encodes exactly the properties exemplified by a given object y; the second that of an abstract object that encodes exactly properties F of the form $[\lambda z\ p]$ constructed out of true propositions p; the third that of an abstract object that encodes all the properties F necessarily implied by a given property G; and the fourth that of an abstract object that encodes exactly the properties F equinumerous with respect to the ordinary objects to a given property G.[17]

Note that in the presence of the definition of identity for objects, each instance of comprehension implies the existence of a unique abstract object satisfying the defining condition ϕ. That in turn entails that, for each description of the form:

$\imath x(A!x \,\&\, \forall F(xF \equiv \phi))$, where ϕ has no free xs

there is a theorem of the form:

$\exists y(y = \imath x(A!x \,\&\, \forall F(xF \equiv \phi)))$

Therefore, these descriptions become canonical in the sense that they are guaranteed to have denotations no matter what the matrix ϕ. Here are examples of such descriptions, which correspond, respectively, with the above examples of comprehension and which are all well-defined:

- The complete concept of y ($\bar{y}$):
  $\bar{y} = \imath x(A!x \,\&\, \forall F(xF \equiv Fy))$

- The actual world (w_α):
  $w_\alpha = \imath x(A!x \,\&\, \forall F(xF \equiv \exists p(p \,\&\, F{=}[\lambda z\ p])))$

- The Platonic Form of G (Φ_G):
  $\Phi_G = \imath x(A!x \,\&\, \forall F(xF \equiv \Box\forall y(Gy \rightarrow Fy)))$

- The Fregean Number of Gs ($\#G$):
  $\#G = \imath x(A!x \,\&\, \forall F(xF \equiv F \approx_E G))$

[17] This is a variant of Frege's notion of equinumerosity, and in Zalta 1999, it is used in the derivation of the Dedekind-Peano axioms for number theory as theorems of object theory.

These examples should give one a good idea of the variety of abstract objects asserted to exist by the theory.

Research on object theory has demonstrated its importance for foundations of mathematics, (foundations of) metaphysics, (foundations of) epistemology, and for regimenting philosophically interesting parts of natural language. The theory can correctly represent sentences and inferences (preserving truth value and validity, respectively) from these subjects that few other axiomatized theory can. Anderson 1993 analyzes the intensional logic and discusses the solution to the paradoxes of object theory. Deutsch 1993 analyzes the way the theory analyzes substitution into propositional attitude contexts. It is established that the theory allows us to reason, and prove fundamental theorems, about possible worlds (Zalta 1983), Platonic Forms (Pelletier and Zalta 2000), the natural numbers (Zalta 1999), Leibnizian concepts (Zalta 2000a), and Fregean extensions (Anderson & Zalta 2004). Moreover, the theory also provides a general foundation for mathematics (Zalta 2000b, Linsky & Zalta 2006). Models of the theory developed by Dana Scott and Peter Aczel were reported in Zalta 1983 and Zalta 1999, respectively.

4. The Problem

We've now outlined the basics of object theory in some detail. To complete our preparations for the statement of the problem, we need to identify some interesting expressions and their properties. Notice that whereas (a) $[\lambda x\ \neg Px]$ and $[\lambda xy\ \forall F(Fx \equiv Fy)]$ are well-formed 1- and 2-place relation terms, respectively, and (b) $[\lambda\ \forall F(Fa \equiv Fb)]$ is a well-defined 0-place relation term (which denotes a proposition), the following expressions are not well-formed λ-expressions:

$[\lambda x\ \exists F(xF \,\&\, \neg Fx)]$

$[\lambda\ \exists F(bF \,\&\, \neg Fb)]$

The formulas in the matrices of the two displayed λ-expressions will play a role in our argument, in Section 4.2, to the conclusion that the logic of object theory cannot be formulated in an FTT. We plan to show that the logic of object theory has to be presented as a logic of *formulas*, and not as a logic of terms. Otherwise, it would miss the inferences involving formulas with encoding subformulas.

To appreciate the argument in Section 4.2, it must be clear why encoding subformulas have been banished from λ-expressions in the language of object theory. Consider what would happen if the following expression were allowed as a term:

$$[\lambda x\, \exists F(xF \,\&\, \neg Fx)]$$

Let us abbreviate this expression as 'K'. If such an expression were a term, we could formulate the following instance of the comprehension principle for abstract objects:

$$\exists x(A!x \,\&\, \forall F(xF \equiv F{=}K))$$

This asserts the existence of an abstract object, call it b, which encodes exactly one property, namely, K. Now to see how a paradox would result, ask the question, does b exemplify K? Suppose b exemplifies K. Then, by the definition of K and β-reduction, there is a property that b encodes but which it fails to exemplify. Since, by definition of b, b encodes only K, b must therefore fail to exemplify K, contrary to hypothesis. So let's suppose b doesn't exemplify K. Then, by the definition of K and β-reduction, it is not the case that there is a property that b encodes and fails to exemplify, i.e., every property b encodes is one it exemplifies. But, by definition of b, b encodes K, and so exemplifies K, again contrary to assumption.

This contradiction is avoided by banishing encoding subformulas from λ-expressions. As a consequence, the formulas with encoding subformulas become expressions that have truth conditions but are not and cannot be converted to terms (i.e., denoting expressions). We developed the logic of object theory as a logic of formulas so that we could represent inferences even involving expressions that were not convertible to terms. And as we shall see in Section 4.1, we can formulate the logic of object theory in RTT as a classical logic of formulas: the classical axioms and rules for propositional logic, predicate logic, and modal S5 can all be preserved in RTT.

4.1 Formulating Object Theory in RTT

Object theory is straightforwardly reformulated in relational type theory, and the logic is essentially preserved, by changing RTT to allow for new (atomic) encoding formulas of the form '$\tau\Delta$' for any term τ of type t and for any predicate Δ of type $\langle t\rangle$. (For example, if 'x' is a term of type t and

'F' is a term of type $\langle t\rangle$, then 'xF' is a formula.) Thus, the representation of object theory in RTT requires the following types:

- i is the type for individuals
- $\langle \alpha_1, \ldots, \alpha_n\rangle$ is the type for relations among entities having types $\alpha_1, \ldots, \alpha_n$

And as a special case when $n = 0$, we have

- $\langle\,\rangle$ (or more simply, p)

as the type for propositions. Then we may type the terms in atomic exemplification formulas of the form $F^n x_1 \ldots x_n$ $(n \geq 0)$ as:

$$F^{\langle \alpha_1,\ldots,\alpha_n\rangle} x_1^{\alpha_1} \ldots x_n^{\alpha_n}$$

So when $n = 0$, F^p is a an exemplification formula.

We also type the terms in atomic encoding formulas of the form xF as:

$$x^{\alpha} F^{\langle\alpha\rangle}$$

An n-place λ-expression $(n \geq 0)$ such as:

$$[\lambda x_1^{\alpha_1} \ldots x_n^{\alpha_n}\, \phi]$$

is an expression of type $\langle \alpha_1, \ldots, \alpha_n\rangle$, and when $n = 0$, $[\lambda\phi]$ is an expression of type p.

The logic of type-theoretic object theory may now be developed in the usual way as a logic of formulas. This is almost straightforward and the details will not be provided here. For the present purposes, we apply the logic assumed for RTT at the end of Section 2.2. To see how this might go in further detail in an enhanced RTT, one may consult Zalta 1983 (Section VI) or Zalta 1988 (Appendix). The main facts to note about such a logic are that inferences are transitions among formulas and that it doesn't matter whether or not the formulas appearing in those inferences are terms, i.e., expressions having denotations. Indeed, the inferences among encoding formulas and formulas that have encoding subformulas are examples of the latter fact, since these are formulas that cannot be construed as denoting terms.

4.2 Formulating Object Theory in FTT

Things are very different when we try to represent type-theoretic object theory in FTT. FTT assumes that every formula is a term and moreover, that every formula can be put inside a λ-expression to form another term. However, in object theory, as we've seen, there are two kinds of formulas: (1) formulas that either are terms or can be put inside λ-expressions to form terms, and (2) formulas that are not terms and cannot be put inside λ-expressions to form terms. Let us call the former 'propositional formulas' and the latter 'non-propositional formulas'. The representation of the inferences involving non-propositional formulas will turn out to be the problem for FTT. Note that in object theory, any propositional formula can be put into a λ-expression, where the λ binds no variables, to form a term that denotes a proposition (relative to assignments to the variables), not a truth-value. For example, as we saw above, in object theory '$Pa \rightarrow \Box Qb$' is a closed propositional formula and can be put into the λ-expression, $[\lambda\ Pa \rightarrow \Box Qb]$. This expression denotes a proposition, not a truth-value.

To see why the logic of non-propositional formulas cannot be represented in FTT, let us consider how to represent various formulas of object theory in FTT. First consider a (propositional) quantified formula of object theory with no encoding subformulas, such as

(A) $\forall x(Rxa)$

To represent (A) in FTT, one must assign R the type (o, ι, ι), and assign both x and a the type ι. By observation 4 at the end of Section 2.2, (A) then becomes, in FTT, an abbreviation of:

(B) $\Pi([\lambda x\ Rxa])$

In (B), $[\lambda x\ Rxa]$ is of type (o, ι), Π is of type (o, o, ι), and thus the term (B) is itself of type o, by clause 2 (functional application) of the definition of terms in FTT (Section 2.2 above).

Now, by contrast, consider a (non-propositional) quantified formula of object theory with encoding subformulas, such as

(C) $\forall x(xP \rightarrow Px)$

To represent this formula in FTT, we have to answer the question, "How should FTT be modified so as to include encoding formulas?"

The simplest modification to FTT that accommodates encoding formulas is to add the name of a new binary function, *Enc*, that maps objects and property-type entities to truth values. We may suppose that the type of *Enc* is $(o, \alpha, (o, \alpha))$, for any type α. So, for example, the simplest formula with *Enc* will be of the form $Enc(x, F)$, where x is of type ι and F is of type (o, ι). The whole formula $Enc(x, F)$ is therefore a term of type o. The formula $Enc(x, F)$ intuitively asserts that x encodes F.

So in translating from object theory to FTT, (C) becomes (C′):

(C′) $\forall x(Enc(x, P) \rightarrow Px)$

In (C′), P appears to be of type (o, ι), and x of type ι, and (C′) itself *should be* of type o. However, it turns out that (C′) is not even well-formed, because given our representation of the universal quantifier, it is an abbreviation of:

(D) $\Pi([\lambda x\ Enc(x, P) \rightarrow Px])$

But we learned from our discussion above that we can't allow encoding subformulas into λ-expressions. For if we could, we could formulate:

(E) $[\lambda x\ \exists F(Enc(x, F)\ \&\ \neg Fx)]$

If this putative λ-expression were well-formed, the paradox discussed above would be derivable.

The moral here is clear. If we introduce encoding formulas into FTT in the natural way, we can't define the universal quantifier in terms of applying the function Π to a λ-expression, since not all formulas in the scope of a quantifier can be converted to λ-expressions. Indeed, the quantifier function Π can't be introduced as a primitive function symbol. Note that if we were to instead introduce the quantifier as a variable binding operator, then there would be closed formulas like $\exists F(Enc(b, F)\ \&\ \neg Fb)$ which would be terms of type o. By λ-abstraction, we could then form terms like $[\lambda x\ \exists F(Enc(x, F)\ \&\ \neg Fx)]$, thereby introducing the property that leads to contradiction. The only way to avoid this would be to somehow stipulate that $\exists F(Enc(b, F)\ \&\ \neg Fb)$ is not a term of any type. But then we would have formulas which aren't terms, contrary to the idea that closed formulas in FTT are just terms of type o.

We've just described one problem with the translation of type-theoretic object theory into FTT. But there is also a second problem, one which arises for expressions of object theory that don't involve quantifiers. Consider the following two formulas:

(F) $Pa \,\&\, \neg Pa$

(G) $aP \,\&\, \neg Pa$

In object theory, both of these are formulas and they are assigned truth conditions by the definition of truth. The first can be converted into the term $[\lambda\, Pa \,\&\, \neg Pa]$; the second cannot be converted into a term.

Now suppose we translate these formulas into FTT with *Enc*. They would become:

(F′) $Pa \,\&\, \neg Pa$

(G′) $Enc(a, P) \,\&\, \neg Pa$

Both expressions would be formulas, i.e., terms of type o. But if we allow (G′) to be a term of type o, then it could be placed inside a λ-expression to form a new term. But this leads to our previous problem, since if '$Enc(a, P)$ & $\neg Pa$' were a term, then no matter how quantification is handled within the system, '$\exists F(Enc(a, F) \,\&\, \neg Fa)$' would become a term, and by λ abstraction, '$[\lambda x\, \exists F(Enc(x, F) \,\&\, \neg Fx)]$' would become a term. This would recreate the λ-expression that leads to paradox.

So we can't even introduce the function *Enc* and assign it a type. For any type we assign to it would turn $Enc(x, F)$ into a term of some type or other. Once it is a term of some type or other, it can be included in expressions to form more complex terms using any of the rules of formation, and in particular, the rule of λ-abstraction. So there is no way to prevent the contradiction-generating λ-expression from being formed.

These two problems now present an obstacle that, we claim, can't be overcome, namely, that FTT cannot represent the logic of the following derivation:

$$aP \,\&\, \neg Pa \vdash \exists F(aF \,\&\, \neg Fa)$$

This is an instance of Existential Generalization that can't be represented in FTT. To represent the derivation in FTT, one would have to have terms on both sides of the derivation sign. But formulas in object theory with encoding subformulas can't be represented as terms of type o in FTT. Thus, a term logic cannot represent those inferences in object theory which relate two formulas with encoding subformulas. Such inferences, however, are valid in the representation of object theory in RTT because the formulas have well-defined truth conditions, and the rules of inference

relate formulas, not terms. This establishes that the formula-based logic of RTT is more general than the term-based logic of FTT.

5. Are There Ways To Dodge This Problem?

We've seen that if one were to represent all object theory formulas as terms, then non-propositional formulas would be embeddable into λ-expressions, leading to paradox. Non-propositional formulas, therefore, can't be terms.

5.1 Would Interpreting FTT Intensionally Help?

Now someone might object that we have assumed the standard extensional interpretation of FTT, in which the terms of type o in FTT denote truth-values, and that by interpreting the terms of type o in FTT intensionally, so as to regard the terms of type o as denoting propositions, the formulas of object theory could be represented as terms denoting propositions. Thus, the *non-propositional* formulas of object theory could get represented as terms denoting propositions.

This suggestion doesn't help, however. If non-propositional formulas were represented as terms denoting propositions, we would be able to abstract over them. Consider, for example, the formula $\exists F(aF \,\&\, \neg Fa)$. If this were to denote a proposition, then we could abstract out the property $[\lambda x\, \exists F(xF \,\&\, \neg Fx)]$. Such a property would yield a contradiction, as described above in Section 4. Thus, it doesn't help to assume an intensional interpretation of FTT.

5.2 Would Free Logic Help?

One might make the following suggestion. Why not reformulate FTT with a free logic? The suggestion here would be to alter the logic of FTT so that it allows for closed terms that don't denote anything. Such terms could be used to represent formulas of object theory with encoding subformulas. If such formulas are represented as non-denoting terms, then we can't abstract out the properties that lead to contradiction. To implement this suggestion in FTT, the logic would have to be adjusted so that the instantiation and substitution of terms is allowed only when one knows that the terms denote.

This suggestion won't work either. We can't make this modification to FTT and still retain or represent the classical logic of object theory. For the suggested modification would require that one allow for formulas (i.e., expressions of type o) that have no denotation. Such formulas would not denote truth-values. But if such expressions don't denote truth-values, then they are neither true nor false and so can't be governed by the principles of classical logic. By contrast, the logic of non-propositional formulas in object theory is classical. For example, it is a theorem of object theory that $\exists FaF \vee \neg\exists FaF$. However, if '$\exists FaF$' were represented in FTT by the term $\exists F Enc(a, F)$ and that term were to denote nothing, then *a fortiori* it wouldn't denote a truth value. The representation of its negation, $\neg\exists F Enc(a, F)$, therefore, would denote nothing, and so wouldn't denote a truth value. If either the representation of $\exists FaF$ or the representation of its negation were to fail to denote, then the representation of their disjunction, $\exists F Enc(a, F) \vee \neg\exists F Enc(a, F)$, wouldn't denote anything. In particular, it wouldn't denote the True and therefore couldn't be valid.

Thus, if we made the logic of FTT free, some logical theorems of object theory that follow from classical propositional logic alone would not be preserved. Object theory preserves the classical principle of bivalence, but its representation in FTT would not. This proposal then fails to offer a faithful representation of object theory.

It does no good to consider amending the free logic for FTT by adding a third truth value, say 'undefined'. The amended proposal still wouldn't preserve the truth values of theorems of object theory. For example, $\exists FaF \vee \neg\exists FaF$ is provable as a theorem of object theory. But on the amended proposal, its representation in FTT would receive the truth-value 'undefined', because at least one disjunct of $\exists F Enc(a, F) \vee \neg\exists F Enc(a, F)$ would be a term with the truth value 'undefined'. That would clearly sever the connection between theoremhood and truth.

5.3 Would it Help to Add a New Primitive Type?

Another suggestion, however, presents itself: why not repair FTT by adding a new primitive type: in addition to primitive types ι (for individuals) and o (for truth-values), why not add the primitive type π (for propositions)? Call the resulting functional type theory FTT′. The idea would then be to represent formulas of object theory in FTT′ as follows:

- Propositional formulas (i.e., formulas that can be put inside λ-expressions to form terms) are to be represented in FTT′ as terms of type π. Terms of type π denote propositions, not truth values.
- Non-propositional formulas (which can't be converted to terms) would be represented in FTT′ as terms of type o. Terms of type o denote truth values, not propositions.

Though this suggestion looks simple and natural enough, the fact is that (a) it doesn't solve the previously-described problem of representing quantified non-propositional formulas of object theory, and (b) it introduces a serious problem concerning the logical connectives.

The addition of a new type π for propositional formulas doesn't solve the problem of representing quantified formulas with encoding subformulas in FTT. Recall that the formula $\forall x(Enc(x, P) \rightarrow Px)$ can't be represented as:

(D) $\Pi([\lambda x\, Enc(x, P) \rightarrow Px])$

because encoding formulas cannot be embedded in λ-expressions. So the addition of a new type doesn't do anything to solve this problem.

But a new and serious problem arises when one considers the logical constants under this new proposal for revising FTT_0. Given the definition of FTT_0, we observed (Section 2.2, observation 3) that the logical connectives are introduced so that:

> Negation ($\neg$) and conditionalization ($\rightarrow$) are simply represented as functions of type (o, o) and type (o, o, o), respectively. So where ϕ is a term of type o, $\neg\phi$ is also a term of type o. Where ϕ, ψ are terms of type o, so is $\rightarrow \phi\, \psi$.

The types of the logical constants, as defined in FTT_0 are functional types applying to terms of type o. Unless we broaden this definition, there would be no way to allow for molecular formulas with subformulas of type π, and so no way to represent the object theoretic formula $Pa \,\&\, Qb$. Thus propositional formulas can't be put into truth-functional combination, nor can one have quantified truth-functional propositional formulas.

Desperate moves might be considered: (a) introduce two different forms of conjunction, negation, conditionalization, quantification, etc., one for propositional formulas and one for non-propositional formulas. This would, of course, require two separate logical systems, one for the

the molecular and quantified formulas of type π and one for the molecular and quantified formulas of type o. If that is what is required, then clearly FTT can't be as general as RTT, since no such bifurcation of the logic is needed to represent object theory in RTT. Nor could one revise the introduction of the connectives and quantifiers so that they become functions solely applying to formulas of type π. For this would prevent us from representing molecular and quantified non-propositional formulas, since on such a suggestion, there would no longer be connectives and quantifiers relating formulas of type o. This latter move abandons all hope of revising FTT so as to solve the problem of representing quantified non-propositional formulas discussed in the middle of Section 4.2.

Since every formula in object theory is governed by the definition of truth and receives truth conditions, propositional formulas are assigned truth conditions. When the truth conditions of a propositional formula obtain, the proposition it denotes has the truth value The True as an extension. Thus, the fundamental problem for representing object theory in functional type theory is that object theory has two kinds of expressions with truth conditions: (1) propositional formulas that denote propositions, and (2) non-propositional formulas that aren't terms and can't be converted to terms. The attempt to translate formulas of object theory which denote propositions into FTT′ fails to represent them in terms of an expression that has the right type: on the one hand, such formulas should be represented as being of type π because they denote propositions, but on the other hand, they should also be represented as being of type o because each propositional formula receives truth conditions and (indirectly) a truth value (namely, the extension of the proposition it denotes). Since type theory doesn't allow unambiguous denoting expressions to be of more than one (primitive) type, one can't classify propositional formulas as both of type π and of type o.[18] FTT′ therefore offers no way to represent propositional formulas of object theory so that they not only denote propositions but also receive truth conditions and a truth value.

[18]Type theory requires that every simple and complex denoting expression be categorized with a unique type, since each primitive denoting expression is then assigned a unique value from the domain of the relevant type, and each complex denoting expression will be assigned a unique semantic value on the basis of the semantic rule that corresponds to the syntactic rule governing the way in which the expression was formed. No denoting expression, whether simple or complex, can be assigned two types without becoming ambiguous. Polymorphic type theories in which a function can take arguments of different types are not relevant here.

5.4 Can We Add a New Mode of Functional Application?

It might be observed, finally, that we extended RTT with a new atomic mode of predication. Could we extend FTT similarly with a new form of functional application?[19] Any answer to this question should be grounded in philosophical conceptions of predication and functional application. We understand what it is for there to be multiple modes of predication; this can be cashed out in terms of multiple ways in which a property might characterize an object. But we don't understand what it would be for there to be multiple modes of functional application. For the idea of a function is grounded in that of a mapping or a correlation, in which some objects are mapped or correlated with some other objects. While there may be multiple ways in which a property can characterize an object, there seems to be only one kind of correlation that functions can achieve. Clearly, the extensional notion of a function, i.e., the one with which everyone is comfortable, seems to admit only one kind of mapping. Though one might argue that an intensional notion of function might admit of kinds, at present, we aren't sure what those kinds might be. Until this suggestion can be fleshed out and given a precise meaning or systematization, we aren't sure how to give an answer to the opening question.

Even if some sense could be made of the notion of two kinds of functional application, there remains the fact that in RTT propositions come for free, as the empty type, whereas in FTT, propositions or truth-values have to be added as a distinguished type. Even if there were a second kind of functional application, the output of the new kind of functional application couldn't be propositions over which abstraction applies. So if classical functional application can map some entities to propositions, the new form of functional application would have to have to output values of some other type. We don't know of any values and types that would not lead back to one of the problems discussed above.

6. Conclusion

We believe our argument establishes that FTT is less general than RTT. We have suggested that functional type theory doesn't have the resources

[19]We'd like to thank Uri Nodelman for raising this possibility. We can't claim to have done justice to the idea, but it is a point worth pursuing in greater depth.

for capturing logics that are easily representable in RTT. The logic of object theory is expressed in a language in which there are formulas that are not terms and that cannot be converted to terms by embedding them in λ-expressions. This logic is representable in RTT but not in FTT. Moreover, our results establish that non-propositional complex formulas involve connectives and quantifiers that cannot be represented as functions. Indeed, we believe we have shown that it is a mistake to eliminate the primitive syntactic category 'formula' in favor of the single primitive category 'term', as is done in FTT. We think it is a mistake to suppose that functional type theory is the only kind of type theory there is, or that RTT is simply a variant of FTT.

There is a deeper philosophical point that can be made, however. It seems clear that some of those who pursue functional type theories (e.g., Church and others inspired by Frege's work) do so in the belief that functional application is more fundamental than predication. They have concluded not only that functions (rather than relations) should be taken as primitive in the foundations of logic, but also that predication can be reduced to to a special case of functional application and naming The True. But such a reduction fails to capture the unique kinds of inferences found in the applications and theorems of object theory, many of which involve a distinctive way of defining and reasoning about mathematical objects. When predication is collapsed to functional application, such forms of reasoning about mathematical objects are lost.

Our work shows that a philosophically and mathematically rich system based on (modes of) predication cannot be reduced to a system that employs only functional application and naming. Relational type theory can serve as the background framework for a type-theoretic version of object theory, but functional type theory cannot. This fact takes on larger significance in the presence of the fact mentioned at the outset, namely, that relations and predication suffice for defining functions and functional application. Recall that Whitehead & Russell showed that (a) $Rxy \,\&\, Rxz \rightarrow y = z$, (b) $R = f$, and (c) $y = \iota z Rxz$ together define the function f as the relation R and define functional application $f(x) = y$ in terms of predication. They thought that relations and predication constitute a better cornerstone for the foundations of logic than functions and functional application. We think they were right. Not only is RTT more general than FTT, but predication is more general than functional application, since the former admits of kinds, one of which can't be reduced

to functional application.

Bibliography

Anderson, C. Anthony, 1993, "Zalta's Intensional Logic", *Philosophical Studies*, 69/2–3: 221–230.

Anderson, D., and E. Zalta, 2004, "Frege, Boolos, and Logical Objects", *Journal of Philosophical Logic*, 33/1 (February): 1–26.

Andrews, P.B., 2002, *An Introduction Mathematical Logic and Type Theory: To Truth Through Proof*, Second Edition, Kluwer Academic Publishers, Dordrecht/Boston/London.

Andrews, P.B., 2006, "Church's Type Theory", *The Stanford Encyclopedia of Philosophy* (Fall 2006 Edition), Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/fall2006/entries/type-theory-church/>.

Andrews, P.B., M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi, 1996, "TPS: A Theorem Proving System for Classical Type Theory", *Journal of Automated Reasoning*, 16: 321–353.

Church, A., 1932, "A Set of Postulates for the Foundation of Logic", *The Annals of Mathematics*, 33/2 (April): 346–366.

Church, A., 1933, "A Set of Postulates for the Foundation of Logic", *The Annals of Mathematics*, 34/4 (October): 839–864.

Church, A., 1940, "A Formulation of the Simple Theory of Types", *The Journal of Symbolic Logic*, 5: 56–68.

Church, A., 1941, *The Calculi of Lambda-Conversion*, Princeton: Princeton University Press.

Deutsch, Harry, 1993, "Zalta on Sense and Substitutivity", *Philosophical Studies*, 69/2–3: 209–220.

Fitelson, B. and E. Zalta, 2007, "Steps Toward a Computational Metaphysics", *Journal of Philosophical Logic*, 36/2 (April): 227–247.

Hylton, P., 1993, "Functions and Propositional Functions in *Principia Mathematica*", in A. Irvine and G. Wedeking (eds.), *Russell and Analytic Philosophy*, Toronto: University of Toronto Press, 342–360.

Kleene, S., 1935, "A theory of positive integers in formal logic", *American Journal of Mathematics*, 57: 153–173, 219–244.

Linsky, B., 2007, "Logical Analysis and Logical Construction", in *The Analytic Turn: Analysis in Early Analytic Philosophy and Phenomenology*, M. Beaney (ed.), London: Routledge, 107–122.

Linsky, B., and E. Zalta, 2006, "What is Neologicism?", *The Bulletin of Symbolic Logic*, 12/1: 60–99.

Mally, E., 1912, *Gegenstandstheoretische Grundlagen der Logik und Logistik*, Leipzig: Barth.

Manzano, M., 1996, *Extensions of First-Order Logic*, Cambridge University Press, Cambridge.

Muskens, R., 1989, "A Relational Formulation of the Theory of Types", *Linguistics and Philosophy*, 12: 325–346.

Pelletier, F.J., and E. Zalta, 2000, "How to Say Goodbye to the Third Man", *Noûs*, 34/2 (June): 165–202.

Russell, B., 1908, "Mathematical Logic as Based on the Theory of Types", *American Journal of Mathematics*, 30: 222–262; reprinted in van Heijenoort 1967, pp. 150–182.

Schönfinkel, M., 1924, "Über die Bausteine der mathematischen Logik," *Mathematische Annalen*, 92: 305–316; translated into English as "On the Building Blocks of Mathematical Logic" in van Heijenoort 1967, 355–366.

van Heijenoort, J., 1967, *From Frege to Gödel. A Source Book in Mathematical Logic: 1879–1931*, Cambridge, MA: Harvard University Press.

Whitehead, A. N., and Russell, B., 1927, *Principia Mathematica* (Volume 1), 2nd edition, Cambridge: Cambridge University Press.

Zalta, E., 1983, *Abstract Objects: An Introduction to Axiomatic Metaphysics*, Dordrecht: D. Reidel.

———, 1988, *Intensional Logic and the Metaphysics of Intentionality*, Cambridge: MIT Press.

———, 1993, "Twenty-Five Basic Theorems in Situation and World Theory", *Journal of Philosophical Logic*, 22: 385–428

———, 1999, "Natural Numbers and Natural Cardinals as Abstract Objects: A Partial Reconstruction of Frege's *Grundgesetze* in Object Theory", *Journal of Philosophical Logic*, 28/6: 619–660.

———, 2000a, "A (Leibnizian) Theory of Concepts", *Philosophiegeschichte und logische Analyse / Logical Analysis and History of Philosophy*, 3: 137–183.

———, 2000b, "Neo-Logicism? An Ontological Reduction of Mathematics to Metaphysics", *Erkenntnis*, 53/1-2: 219–265.