
RELATING THEORIES OF THE $\lambda$-CALCULUS

Dana S. Scott

Wessex College
Oxford

Dedicated to Professor H. B. Curry on the occasion of his 80th Birthday

Mathematical theories arise for many different reasons, sometimes in connection with specific applications and often owing to accidental inspiration. From time to time we ought to ask ourselves concerning our theories where should they have come from; usually the answer will have little to do with the exact historical development. The $\lambda$-calculus is, I feel, a case in point. In Scott (1980), in the Kleene Festschrift, I made up a story of where the theory of type-free $\lambda$-calculus could have come from. Any number of people who heard my lecture and read the manuscript were cross with me. They said "But it didn't develop that way! And besides we doubt it ever would have." But this reaction misses the point of my story. I shall not, however, repeat the earlier story here, for the point of the present paper is different. For those people who do not like to discuss philosophy - even Philosophy of Mathematics - my remarks here can be taken as a suggestion of how to group diverse models of $\lambda$-calculus rather uniformly under a general scheme. The scheme is by now rather well known and not at all original with me. What I hope can be regarded as a useful contribution is my putting of the ideas in a certain order. As I consider the order to be a natural one, I feel there is philo-
sophical significance to my activity; but I should not want to force this view on anyone.

1. THEORIES OF FUNCTIONS.

Everyone agrees that \( \lambda \)-calculus is a theory of functions. But we must ask: "What kind of a theory?" And also: "Have we got the best theory?" Personally, I think we should also inquire: "How does it relate to other theories?" I certainly find many discussions far too silent on this last issue.

Well, what other theories are there? Certainly \textit{set theory} comes to mind at once, and no set theory would be worth its salt if it did not provide a theory of functions. Let us not try to catalogue the various known theories here but look at a theory in the style of Zermelo — and we do not have even to be too specific, since in any case such a theory is very standard. What is "unsatisfactory" about Zermelo's theory is the limitation-of-size view of sets: any one set \( A \) is extremely small compared to the size of \( V \), the class or universe of all sets.

Thus, functions \( f: A \rightarrow B \) mapping one set \( A \) into another set \( B \) tell us very little about operations on all sets, maps on \( V \) into \( V \). We therefore have an urge to "improve" our set theory by constructing a \textit{class theory}. Sets are elements \( A \in V \); while classes are subcollections \( B \subseteq V \). As \( V \) is (by the usual assumptions) so highly closed under so many operations, we have no difficulty in construing certain classes as maps \( F: V \rightarrow V \).

For example for all \( x \in V \) we could have \( F(x) = \{x\} \) or \( F(x) = A \times X \) (where \( A \) is a fixed set).

The passage from sets to classes is a familiar and useful move in the formalization of the theory: many things can be done generally for classes and then specialized to sets. And having a notation for functions defined on all sets is in many cases a great advantage. But wait. What about operations on classes? What should we say about them? Given any two classes \( A \) and \( B \), we can form their union, \( A \cup B \). The operation, \( U: V \times V \rightarrow V \), of union of sets does not directly apply to classes even though there is a connection. Do we also want a theory of class operations? Do we have to go to hyperclasses (classes of classes)? Is there any end to this expansion?

\textit{[An Aside: The story of Scott (1980) was meant to suggest one answer — the one known to Plotkin (1972).} Namely, we consider only "continuous" class operations. These are objects \( F \) such that \( F(X) \) is defined for every class \( X \subseteq V \) and \( F(X) \) is a class, too. Moreover \( F \) should satisfy:

1. \( X \subseteq Y \) always implies \( F(X) \subseteq F(Y) \);
2. Whenever \( A \subseteq F(X) \) and \( A \) is a set, then \( A \subseteq F(B) \) for some set \( B \subseteq X \).

We do not have time to discuss the justification of the word "continuous" here; suffice it to say that conditions (1) and (2) are not as strict as they at first might seem. Every ordinary map \( f: V \rightarrow V \) determines a continuous class operator by the definition:

\[ F(X) = \{f(x) \mid x \in X\}. \]

Furthermore, \( F \) determines \( f \), for we have:

\[ y = f(x) \text{ iff } \{y\} = F(\{x\}), \]

for all \( x, y \in V \). In a suitable sense, then, nothing has been lost; but what has been gained?

The reply is that \textit{continuous class operators can be identified with classes}. We could write, for instance:

\[ F = \{(A, B) \mid A, B \in V \land A \subseteq F(B)\}, \]

where, say:

\[ (A, B) = \{\{A\}, \{A, B\}\}. \]

More in harmony with Scott (1980) would be:

\[ F = \{(a, B) \mid a, B \in V \land a \in F(B)\}. \]
Either trick reduces operator theory to class theory—in the continuous case. And the same trick could be carried over to other kinds of set theory (e.g. Quine’s). What we know is that operator theory gives a model for $\lambda$-calculus; it is a quite elementary model, too.

Nice as this connection is, it is not the topic of the present paper: we do not want to make $\lambda$-calculus depend on set theory, since then we have still to explain where set theory comes from. But the connection should be borne in mind.

Perhaps set theory brings in too many extraneous issues. $V$, after all, is a massive object closed under all manner of strange operations. What we are probably seeking is a "purer" view of functions: a theory of functions in themselves, not a theory of functions derived from sets. What, then, is a pure theory of functions? Answer: category theory.

General category theory is a very pure theory: it is the milk-and-water theory of functions under composition. This composition operation is associative and possesses neutral elements (compositions of zero terms). That is about all you can say about it except to stress that it is also a rather bland theory of types. Every function $f$ has a (unique) domain and codomain, and we write:

$$f : \text{dom } f \to \text{cod } f$$

Every possible domain is a codomain (and conversely), because if $A$ is such, then

$$\text{dom } l_A = A = \text{cod } l_A,$$

where $l_A$ is the neutral element of type $A$.

[If we want to be especially parsimonious in entities, we can even write $l_A = A$, because each of $l_A$ and $A$ uniquely determines the other.]

The point of distinguishing domains and codomains is not only do they specify the type of $f$, but a composition $g \circ f$ is defined if, and only if, $\text{dom } g = \text{cod } f$. And then $\text{dom } (g \circ f) = \text{dom } f$ and $\text{cod } (g \circ f) = \text{cod } g$. We usually write this as a "rule of inference":

$$\frac{f : A \to B \quad g : B \to C}{g \circ f : A \to C}$$

with the understanding that the typing of $f \circ g$ can only be obtained by such an application of the rule. The types, then, are invoked just to type functions, and the only theory involved is that of the "transition" of types under composition.

Sets (and set-theoretical mappings) do of course form a category; category theory is meant to be more general than set theory. We should construe the function entities here as triples of sets $(A, f, B)$ where

$$f \subseteq A \times B \land \forall x \in A \exists y \in B. (x, y) \in f.$$  

The definition of composition is obvious. Sets, in this way, give us only one special example of a category.

I beg forgiveness of the reader for boring him. All of this is well known to the moderately awake undergraduate in mathematics. Indeed, that is the point: there is plenty of evidence now that category theory is a natural and useful theory of functions. I do not have to rehearse the examples as they can be found in any number of books (e.g. Mac Lane [1971]).

There is a rather important logical point to stress, however—important for anyone who has thought about $\lambda$-calculus models. Category theory is very extensional. We assume as axioms the equations:

$$l_A \circ f = f \circ l_B = f$$

and

$$h \circ (g \circ f) = (h \circ g) \circ f,$$

provided $A = \text{dom } f$ and $B = \text{cod } f$ and the double compositions are defined. These are functional equations, and they say that two functions defined in different ways are in fact identical.
Furthermore, identical things can everywhere replace one another.

This point about extensionality may not seem exciting or important, but the logician should remember that, in certain intensional theories of functions, "obvious" definitions will not provide categories. We shall return to this point later.

But is category theory the long-sought answer? No, no, not at all. Category theory pure provides nothing explicitly aside from identity functions - and they occur only if we have some possible domain. We do get compositions if we have the necessary terms. Thus, as it stands, category theory has no existential import. (It was not meant to.) Set theory has "too much" existential import. (It was meant to.) What we seek is the middle way - and an argument that the middle way is natural and general.

There is no need to build up unnecessary suspense: the middle way is the theory of the (so-called) cartesian closed categories. Fortunately Lambek has written extensively about the theory, and I can refer the reader to his papers for further details; I also am happy to acknowledge his writings as helping me understand what is going on. If we remark that his paper in this volume is called "From $\lambda$-calculus to cartesian closed categories", we might say that my present paper ought to be called "From cartesian closed categories to $\lambda$-calculus." I am trying to find out where $\lambda$-calculus should come from, and the fact that the notion of a cartesian closed category (c.c.c) is a late developing one (Eilenberg & Kelly (1966)), is not relevant to the argument: I shall try to explain in my own words in the next section why we should look to it first.

2. A THEORY OF TYPES

I say "a theory", because there are many possible theories; indeed pure category theory is one of the theories. Its weakness lies in the fact that we are given no construction principles, no way of making new types from old. From the point of view of logic what should we expect? What more do we want to say beyond relations between types which hold when a mapping statement $f: A \to B$ obtains.

An immediate question that must come to anyone's mind concerns the arity of functions. The usual way of reading a mapping statement is to take it as a statement about one-place functions, and the $*$ of composition is the composition of one-place functions. This seems very restricted.

People have suggested generalizing categories to multi-place functions with concomitant compositions (cf. e.g. the book Szaabo (1978)), but it does not seem the neatest solution. Much easier is to assume that the category has cartesian products - and more specifically particular representatives of the product domains are chosen. As a special case we will know what the cartesian power $A^n$ is for each $n=0,1,2,...$, and $n$-ary functions are then maps $f: A^n \to B$. Not much of a surprise.

We have to take care, however. In the first place, a given category may not have cartesian products (it fails to have enough types). Even if it does, the maps allowed may be too restricted - for logical purposes. Take the category of groups and homomorphisms, for example. The required products exist. A map $f: A^n \to B$ in this category has to be a group homomorphism, naturally. Suppose two maps $u,v: A \to B$ were given. Intuitively we think in terms of elements and that we are mapping $x \mapsto u(x)$ and $y \mapsto v(y)$. The pointwise group product of the maps, namely, $x,y \mapsto u(x) \cdot v(y)$ is a very nice map $g: A^2 \to B$ in the ordinary sense - but unless the group $B$ is abelian, $g$ is not a homomorphism. It is a "logical" map but not an "algebraic" map. Pure category theory applies to many algebraic situations (as everyone knows that is why it is a good theory), but not all categories are "logical" even if they have products.

In the example of groups, what was "missing" was the group
multiplication $\mu : B^2 \to B$ as a map in the category. (Inverse
is missing as well, since it reverses order.) There is an
interesting theory of algebraic theories that address the ques-
tion of the proper categorial construction of categories of al-
gebras, but I do not think we should invoke that theory here.

The precise description of products is as follows. We as-
sume our category has a special domain $1$ (the empty product, so
$A^0 = B^0 = 1$), and for each domain $A$ a special map $0_A : A \to 1$.
(The domain $1$, intuitively, has just one "element"). Moreover
the rule about maps is that $0_A$ is unique; that is whenever
$f : A \to 1$, then we have the equation

$$f = 0_A.$$

Concerning binary products, we have for any two domains $A$
and $B$ a special choice of a domain $A \times B$ (and so $A^{n+1} = A^n \times A$),
and special maps

$$p_{AB} : A \times B \to A$$

$$q_{AB} : A \times B \to B$$

But the mere existence of "projections" does not characterize
$A \times B$ as a product. We have to assume that there is a chosen
pairing operation $\langle f, g \rangle$ on maps such that types are assigned
by the rule:

$$f : C \to A \quad g : C \to B$$

$$\langle f, g \rangle : C \to A \times B$$

Moreover, provided $f$ and $g$ are as above and $h : C \to A \times B$ we
have to assume

$$p_{AB} \circ \langle f, g \rangle = f$$

$$q_{AB} \circ \langle f, g \rangle = g$$

$$\langle p_{AB} \circ h, q_{AB} \circ h \rangle = h.$$

that is to say, there is an explicit one-one correspondence be-
tween the pairs of maps $f, g$ and the maps $h$ into the product.
This all now makes $A \times B$ well behaved within the category.

So much for a theory of tuples and multiary maps. But we
still want a theory of functions: a category allows us to talk
of selected functions, while we would want various equations to
have a force relating to arbitrary functions. The answer to
this desire is function spaces as explicit domains in the cate-
gory. Given $A$ and $B$ we want to form $(A \to B)$ as a domain in its
own right; if so, there are many maps that have to be set down
to make the function space behave. (And here we must defi-
nitely leave the category of groups.)

In the first place there has to be an evaluation map

$$e_{BC} : (B \to C) \times B \to C$$

with the intuitive interpretation that it is the map $f, x \mapsto f(x)$. In the second place there has to be a
map for shifting around variables; more precisely, suppose
$h : A \times B \to C$ is a map with two arguments. In an evaluation
$h(x, y)$, we can think of holding y constant and regarding $h(x, y)$
as a function of $x$. We need a name for this function – and for
the correspondence with possible values of $x$. We write

$$\lambda_{AB} h : A \to (B \to C)$$

so that the function we were thinking of – given $x$ – was

$$(\lambda_{AB} h)(x).$$

But all this function-value notation is not cate-
gorical notation; what we have to say is that there is a one-
one correspondence via $\lambda$ between maps $h : A \times B \to C$ and maps

$$k : A \to (B \to C).$$

This comes down to these two equations:

$$e \circ \langle \lambda h \rangle \circ p, q = h,$n and

$$h \circ (e \circ k \circ p, q) = k.$$

(It is necessary to have subscripts here: $\lambda_{ABC}$, $p_{AB}$, $q_{AB}$, and
$e_{BC}$; but we leave them off when there is no ambiguity.)

The notation is now wholly categorical (and mostly unread-
able). Category theorists put the whole thing (that is, the
definition of a c.c.c., which is what we have just given) into
the language of functors, which has a lot of sense. But if you have never seen any abstract category theory before, it is really rather too abstract. The idea of a c.c.c. as a system of types is, I think, reasonably simple. Each c.c.c. represents a theory of functions. The maps in the category are certain special functions that are used to express the relations between the types (the domains of the category). In order to be able to deal with multiary functions, we assume we can form (and analyze) products. In order to be able to work with transformations of arbitrary functions ("arbitrary" within the theory), we assume we can form function spaces: this is where "higher" types enter the theory, as in the sequence of domains:

\[ A, (A \to A), ((A \to A) \to A), (((A \to A) \to A) \to A), \ldots \]

To be able really to view these domains as function spaces, certain operations, \(\varepsilon\) and \(\Lambda\), with characteristic equations have to be laid down.

If a c.c.c. is a theory of functions (and we include here higher-type functions), then the theory of c.c.c.'s is the theory of types of the title of this section. It is only one such theory. "Bigger" theories could be obtained by demanding more types: for example we could axiomatize coproducts (disjoint sums), \(\emptyset\), and \(A + B\). We could demand infinite products and coproducts. We could throw in a type \(\Omega\) of "propositions" so that higher types like \(A^\Omega\) correspond to n-ary predicates. This gets into topos theory (as in Johnstone (1977) or Goldblatt (1979) - just to name two recent texts). But the bigger the theory, the more involved, and full definitions at this point would not help this discussion very much.

We could also look for "smaller" theories. Some examples - of a rather highly formal nature - can be found in Szabo (1978) with an indication of the algebraic interest of these other type theories. However, a c.c.c. is rather more "logical" and good as a middle ground; further Lambek has explained the logical interest; there is a perfect correspondence between c.c.c.'s and (extensional) typed \(\lambda\)-calculi. The reader can turn to Lambek's paper for details and references.

Roughly put, when we formally define the typed \(\lambda\)-language (with types in the given c.c.c.), then if \(\tau\) is a typed \(\lambda\)-expression with free variables of types \(A_0, A_1, \ldots, A_{n-1}\), we can define the "meaning" of \(\tau\) as a map

\[ [[\tau]] : A_0 \times A_1 \times \ldots \times A_{n-1} \to B, \]

where \(B\) is the type of \(\tau\). For example if \(u\) is a variable of type \((B \to C)\) and \(v\) a variable of type \(B\), then

\[ [[u(v)]] : (B \to C) \times B \to C, \]

and in fact \(=[[u(v)]] = \varepsilon_{BC}\). Also if \(x\) is the variable in \(\tau\) of type \(A_{n-1}\), then

\[ [[\lambda x \cdot \tau]] : A_0 \times \ldots \times A_{n-2} \to (A_{n-1} \to B), \]

and in fact \(=[[\lambda x \cdot \tau]] = \Lambda[[\tau]]\). (Warning: for other of the variables that are not the last mentioned, it is not so easy to write down the answer: some permutations of the products have to be introduced.)

The two characteristic equations for \(\varepsilon\) and \(\Lambda\) in the axioms for a c.c.c. have very familiar translations:

\[ (\lambda y \cdot h(x,y))(y') = h(x,y') \]

and

\[ \lambda y \cdot k(x)(y) = k(x), \]

where the type of \(x\) is \(A\), the type of \(y\) and of \(y'\) is \(B\). (That is to say, the \(\langle \cdot, \rangle\)-meaning of the two sides of the equation is the same map in the category.) Of course this all has to be defined more rigorously, but I hope I have conveyed the main part of the idea of Lambek's correspondence. A typed \(\lambda\)-calculus (with pairs, products, and function spaces) is just another notation for a c.c.c.
No, we have to be more specific than that. Take a c.c.c. How does it correspond to a theory (of functions)? The domains of the category are the types of the theory, and they are structured by the \( 1 \), \((A \times B)\), \((A \to B)\) operations on types. Things like \( =, 0, \#, q, <, , >, \langle, \rangle, \varepsilon \), \( \Lambda \) stand for logical constants (or operators) with type subscripts as needed. Maps \( f : A \to B \) of the category stand for the non-logical constants of the theory. The equations \( f = g \) between maps are the assertions of the theory. The logical axioms are those special equations common to all c.c.c.'s - the other equations are those that just happen to work out in the category. From this point of view the theory has no free variables: all assertions are written with constant terms. Equations with free variables can be construed as functional equations (by a heavy use of \( \lambda \)).

Conversely, a more conventional typed \( \lambda \)-calculus is an equational theory with both the familiar logical axioms as well as with non-logical axioms as desired. The equations can involve free variables. Aside from the usual deduction rules for equality, we must employ the extensionality rule

\[
\frac{\tau = \sigma}{\lambda x . \tau = \lambda x . \sigma}
\]

A category is formed from the types (which are given as closed under \( 1 \), \((A \times B)\), \((A \to B)\)). The terms all have unambiguous types, and they are divided into equivalence classes by the theory. As the maps of the category we take the equivalence classes \([\lambda x . \tau]\) where the term \( \tau \) of type \( B \) has at most the variable \( x \) of type \( A \) free, and we write \([\lambda x . \tau] : A \to B\). Of course

\[
1_A = [\lambda x . x], \quad [\lambda y . \sigma] \circ [\lambda x . \tau] = [\lambda x . \sigma(\tau/y)]
\]

where \( \tau \) is substituted for \( y \) in \( \sigma \). We must verify that a category is obtained - using the laws of \( \lambda \)-calculus. And we must see that if we go back again to a \( \lambda \)-calculus from the category we have essentially the same theory.

A c.c.c. (or typed \( \lambda \)-calculus - with non-logical axioms) is a satisfactory (extensional) theory of functions because all we have built into the theory is the idea of the product and the function space. The axioms set down are just those needed to make this structuring explicit.

The reason that category theory is a convenient way to formalize this definition is that starting from the especially elementary concept of maps under composition, we can see that we have done nothing more than close up under products and function spaces. \( \lambda \)-calculus, then, becomes mostly a notational device for setting down our functional equations. At least for typed \( \lambda \)-calculus, we can see in this way that it is harmless.

The typed \( \lambda \)-calculus is even more harmless than these last remarks suggested. By the well-known Yoneda embedding, one can prove that an arbitrary (small) category has a full and faithful embedding into a c.c.c. This means that starting with a given category and its maps, there is a precise sense in which it is consistent to close up under products and function spaces. No new maps are added to the given category; no new equations between the given maps are imposed by the adjunction of higher types. One can even more than this about relative consistency, but the remark is best deferred to Section 4, where references to the proof are provided.

A final remark must be added to this section to clear up a possible confusion between theories and models.

Up to this point we have been talking about theories. In many systems of logic models can be described by theories: every model has a "diagram" involving constants for all the "elements" of the model and taking as axioms all statements in
the language "true" about the model. It depends on the nature of the logic how hard it is to show that every "consistent" theory has a model.

In the case of a c.c.c., a domain $\mathfrak{A}$ could be said to have an "element" if there is a map $a : \mathfrak{I} \to \mathfrak{A}$. The question is: are there enough elements? Suppose $f, g : \mathfrak{A} \to \mathfrak{B}$ are two maps. If $a : \mathfrak{I} \to \mathfrak{A}$, then $f \circ a : \mathfrak{I} \to \mathfrak{B}$; so in a certain sense maps in the category behave as functions on elements. (This is not an original suggestion but is one well known in category theory.) It is natural to ask whether, if $f \circ a = g \circ a$ for all $a : \mathfrak{I} \to \mathfrak{A}$, then $f = g$. If this is true in a c.c.c., then it is said to have "enough" elements or to be concrete. In case it is concrete, domains can be identified with sets, maps with functions, products $\mathfrak{A} \times \mathfrak{B}$ in the category with the corresponding cartesian product of the sets (ask: which $c : \mathfrak{I} \to \mathfrak{A} \times \mathfrak{B}$?), and function spaces $(\mathfrak{A} \to \mathfrak{B})$ with spaces of actual functions (because there is a one-one correspondence between maps $f : \mathfrak{A} \to \mathfrak{B}$ and elements $e : \mathfrak{I} \to (\mathfrak{A} \to \mathfrak{B})$).

For a theory in the form of a c.c.c., to ask whether it has a (non-trivial) model is to ask whether it can be expanded to a concrete c.c.c. by the adjunction of elements (and other maps and additional equations, but no new domains) which is non-trivial in the sense of not making all domains isomorphic to $\mathfrak{I}$.

An answer — though perhaps a rather formal one — is supplied by the method of adjunction of indeterminates $x : \mathfrak{I} \to \mathfrak{A}$ presented in Lambek's paper (this volume). We just have to adjoin infinitely many for each domain, one after the other. Each polynomial involves only finitely many indeterminates. But the results stated by Lambek (esp. Corollary to Theorem 2) show us at once that this expanded category is concrete. The idea is really just like the idea of having "free algebras" for any equational theory. (In $\lambda$-calculus an algebraic equation that is regarded as universally quantified, say $x + y = y + x$, is replaced by the functional equation

$$\lambda x \lambda y . \ x + y = \lambda x \lambda y . \ y + x.$$ More thoughts on concreteness will be brought out in Section 4.

That is the (easy) passage from theories to (certain) models. But remember, a theory is not a model: the maps in a given c.c.c. are not concrete maps, they are just the definable maps in the language of the theory, and the equations between them are the "theorems" of the theory. It is no surprise that a given theory may not have enough definable elements: we may need to expand the stock of elements in order to have a model.

For a c.c.c. we find we can. So far, so good; and this is the (known) story of typed $\lambda$-calculus.

3. "TYPE-FREE" DOMAINS

In the paper of Lambek, the analogy between typed and type-free is illustrated (in the obvious way), but no real connection or relation is established. This we shall now do, and the relationship will be deepened in the next section.

In the first place, we shall only consider the $\lambda$-calculus (or $\lambda K$-calculus) and not the $\lambda \eta$-calculus; the latter can be regarded as a special case. What is needed is a notion of domain appropriate to the interpretation of the "type-free" calculus.

In a category, a retraction between two domains $\mathfrak{A}$ and $\mathfrak{B}$ is a pair of maps $i : \mathfrak{A} \to \mathfrak{B}$ and $j : \mathfrak{B} \to \mathfrak{A}$ where $j \circ i = 1_{\mathfrak{A}}$. Respect $\mathfrak{A}$ as the "smaller" domain; it is injected into $\mathfrak{B}$, and $\mathfrak{B}$ is subjectively mapped onto $\mathfrak{A}$. The notion shares qualities, then, of $\mathfrak{A}$ being both a subspace of $\mathfrak{B}$ and at the same time a quotient. But the injection and surjection have to be related.
Now, suppose that in a cartesian closed category a domain $U$ satisfies the condition that the function space $(U \to U)$ is a retract of the domain $U$ itself. (This is always so for $U = 1$, but we seek non-trivial examples.) Let the retraction maps be $i : (U \to U) \to U$ and $j : U \to (U \to U)$. Then $U$ (as it sits in its category) gives us an interpretation of the type-free calculus, which we now explain.

Let the type-free terms be constructed in the usual way from variables $x,y,z,\ldots$ by means of application and $\lambda$-abstraction. Think of all variables as being of type $U$ and define a translation $\tau^*$ from untyped terms to typed terms so that

- $x^* = x$,
- $(\tau(\sigma))^* = j(\tau^*)(\sigma^*)$,
- $(\lambda x . \tau)^* = i(\lambda x . \tau^*)$.

We intend this in such a way that $\tau^*$ is always of type $U$. The type-free theory (determined by the category, the domain $U$, and by the choice of $i$ and $j$) has as its assertions exactly those equations $\tau = \sigma$ where $\tau^* = \sigma^*$ in the category. The theory satisfies $(\alpha)$, $(\beta)$-conversion, all the rules of equality, and the rule $(\xi)$: from $\tau = \sigma$ to deduce $\lambda x . \tau = \lambda x . \sigma$. This much is surely obvious to anyone reading Lambek's paper.

What I would like to point out here is the converse: given any type-free theory, there is a c.c.c. and a domain $U$ (with a suitable retraction pair $i,j$) so that the above interpretation gives exactly the same type-free theory. Consequently, nothing is lost in considering type-free theories just as special parts of typed theories. I do not find this result mentioned by Lambek.

The proof is elementary. Let the domains for the category be the $\lambda$-terms $A$, without free variables, for which we can prove in the theory:

$$A = \lambda x . A(A(x)).$$

The maps $f : A \to B$ are terms $f$ without free variables for which we can prove

$$f = \lambda x . B(f(A(x))).$$

The equations between maps are the equations we can prove in the theory. [Actually, it might be better to construe maps as triples $(A, f, B)$, but never mind.] It is not hard to show that this is a category where

$$1_A = A,$$

and

$$f \circ g = \lambda x . f(g(x)).$$

[More properly spoken, the maps should be equivalence classes of terms based on the equations of theory, but never mind.]

To show this construction gives a c.c.c. we need to define:

$$A \times B = \lambda u z . z(A(u(\lambda x y . x))(B(u(\lambda x y . y))),$$

where

$$p_{AB} = \lambda u . (A \times B)(u)(\lambda x y . x),$$

and if $f : C \to A$ and $g : C \to B$, then

$$<f,g> = \lambda t z . z(f(t))(g(t)).$$

All of this is based on the familiar pairing functions of $\lambda$-calculus.

For function spaces, we define:

$$(A \to B) = \lambda f. B \circ f \circ A$$

where

$$e_{BC} = \lambda u . C(u(\lambda x y . x))(B(u(\lambda x y . y))),$$

and

$${A}_{ABC} h = \lambda x y . h(\lambda z . z(x)(y)),$$

provided $h : (A \times B) \to C.$
There are a jolly lot of equations to verify, but the work is all straightforward conversion. The method of retracts as a c.c.c. has in any case been exposed before with respect to the Pu model in Scott (1976). Note here, however, we are to verify the required equations in a theory (not a model) making use of nothing but the "logical" axioms of \(\lambda\)-calculus.

It remains to identify the domain \(U\) in the constructed category. We define:

\[
U = \lambda x. x
\]

Clearly

\[
U = \lambda x. U(U(x)) = U \circ U.
\]

Note that every \(A\) in the category is a retract of \(U\); indeed, for retraction define:

\[
A : A \to U \quad \text{and} \quad A : U \to A\quad \text{and} \quad A \circ A = A = 1_A.
\]

We thus speak of these \(A\)'s also as retracts. We can write:

\[
(U \to U) = \lambda f \lambda x. f(x),
\]

and it is thus easy to verify now that \((U \to U)\) is a retract of \(U\). As \(U\) is in fact the identity function, the reinterpretation via \(U\) of the type-free calculus will obviously translate every term into itself.

I just note in writing down the definition of the c.c.c., I forgot to define \(1\) - because it is so dull, I suppose! For this we have to map everything onto a constant:

\[
1 = \lambda u \lambda x. x, \quad \text{and} \quad 0 = \text{false}.
\]

I think the calculations suggested provide an argument that type-free \(\lambda\)-calculus takes second place to typed \(\lambda\)-calculus - foundationally speaking. Type-free domains are special kinds of types. As I have said before in other writings, to get \((U \to U)\) inside \(U\), we have to pass to an infinite type. I thought this was made very clear in the so-called \(D_n\)-construction. The category of continuous lattices and continuous functions is a c.c.c. Starting with any domain \(D_0\), in that category the sequence of types \(D_n\) where \(D_{n+1} = (D_n \to D_n)\) has a certain limit \(D_\infty\) with \(D_0\) (and all the \(D_n\)'s) as retracts, and with \((D_\infty \to D_\infty)\) not only a retract but an isomorph of \(D_\infty\). That is one choice of an \(U\), and I showed many variations are possible for other type-free domains \(U\) in this one category.

We hasten to note that in the c.c.c. of sets and arbitrary functions, a non-trivial domain \(U\) with \((U \to U)\) a retract is impossible (by cardinality considerations). This means that not all c.c.c. lead directly to interpretations of the type-free theory. Hence, we must conclude, the typed theory is the more general one, and the prior one.

Such a conclusion will not be welcome, however. The type-free theory from our experience seems general enough. Even though we have shown two good ways of relating the two kinds of theories, we would like something more. We do not want just some c.c.c. related to a given type-free theory, but we would like to find a relation that achieved any desired c.c.c., provided we cook up the type-free one properly. This problem is the topic of the next section.

Before we turn to this new relationship, a word about models of the type-free calculus would be to the point. There is considerable discussion of the notion of a model in Hindley and Longo (1980) and Barendregt (1980) (where other references are given). We should state how this all fits in with the present view.

When presenting a theory in the usual \(\lambda\)-notation, free variables are permitted as well as full use of the rule (\(\xi\)). But, when thinking of elements (relative to a theory) only terms (better: equivalence classes of terms) without free variables
should be considered. As is known from many examples, there
may not be enough of them. This can of course be so even if we
allow in our language many non-logical constants. What does
"enough" mean? Well, if \( f \) and \( g \) are closed terms, it may be
that \( f(a) = g(a) \) is provable for all closed \( a \), but \( f = g \) is not
provable. The fact that this happens for some theories should
come as no surprise. (For the explicit examples consult Baren-
dregt (1980).)

The remedy is to adjoin indeterminates (constants without
new axioms) until "enough" is reached. (A proof is also found
in Barendregt (1980).) As with the typed calculus, every the-
ory has a model which satisfies exactly the same equations as
are provable in the theory (one might call it a conservative
model).

The notion of a \( \lambda \)-model has not struck people as quite sat-
sactory because the extensionality principle in the "enough"
clause is not very algebraic. A suggestion of mine is mention-
ed in the cited references, but I think it would be useful to
recast the idea in the light of the present discussion.

In typed \( \lambda \)-calculus, the categorical formulation is one way
of eliminating all use of variables. In type-free \( \lambda \)-calculus,
the usual plan is to use the combinators - and the plan leads
to awfully long formulae. Let us not try to give a variable-
free formulation, but talk in terms of first-order models.
What is unalgebraic in the model definition is the \( \lambda \)-operator,
since a bound variable is of the essence of the use of \( \lambda \). So
let us replace \( \lambda \) by the combinators in the usual way. We take
\( S \) and \( K \) as primitive, and a \( \lambda \)-model is (at least), a structure
of the form \( \langle U, \cdot (\cdot), S, K \rangle \), with a domain, a binary operation,
and two distinguished constants. The problem is: what are the
axioms? Clearly we want:

\[
\begin{align*}
K(x)(y) &= x, \\
S(u)(v)(x) &= u(x)(v(x)),
\end{align*}
\]

as usual. But these are not sufficient to express extensional-
ity, which in \( \lambda \)-notation reads:

\[
\forall x. \tau = \sigma + \lambda x. \tau = \lambda x. \sigma
\]

If we convert out the variable \( x \), we are tempted to write:

\[
\forall x. f(x) = g(x) + f = g.
\]

But this is too strong. (It corresponds to \( U = (U \to U) \) rather
than the weaker: \( (U \to U) \) is a retract of \( U \).) If we wrote:

\[
\forall x. f(x) = g(x) + \lambda x. f(x) = \lambda x. g(x),
\]

the statement would at least be correct - even if containing
the unwanted \( \lambda \). Well, we just have to define this \( \lambda \) in terms
of \( S \) and \( K \). Introduce the standard definitions:

\[
\begin{align*}
I &= S(K)(K) \\
B &= S(K(S))(K).
\end{align*}
\]

Then (with \( \lambda \)-notation)

\[
\lambda x. f(x) = \beta(I)(f).
\]

So the desired axiom now reads

\[
(\forall x. f(x) = g(x) + B(I)(f) = B(I)(g)
\]

We are not quite done, however. We want \( S \) and \( K \) to corre-
respond to \( \lambda \)-expressions (eventually), so we need an axiom which
makes them suitably unique. Now we note intuitively that

\[
\lambda x_0 \lambda x_1 \ldots \lambda x_{n-1}. f(x_0)(x_1) \ldots (x_{n-1}) = B^n(I)(f).
\]

Thus, what we need to say is:

\[
\begin{align*}
S &= B(B(B(I)))(S), \\
K &= B(B(I))(K)
\end{align*}
\]
To see that (\(*\)), (\(**\)), and (\(***\)) are adequate, we note first that

\[ B(I)(f)(x) = f(x) \]

by (\(*\)). From (\(**\)) it then follows that

\[ B(I)(B(I)(f)) = B(I)(f) \]

This means we can reformulate (\(**\)) as:

(\(***_{1}\)) \quad f = B(I)(f) \land g = B(I)(g) \land \forall x. f(x) = g(x) \rightarrow f = g.

[This does not seem to be equivalent to (\(**\)) unless we have the equation about \(B(I)(B(I)(f))\) just noted - the retraction equation.] We now generalize (\(***_{1}\)) to \(n\) variables:

(\(***_{n}\)) \quad f = B^{n}(I)(f) \land g = B^{n}(I)(g) \land \forall x_{0}, x_{1}, \ldots, x_{n-1}. f(x_{0})(x_{1}) \cdots (x_{n-1}) = g(x_{0})(x_{1}) \cdots (x_{n-1}) \rightarrow f = g.

If we prove this, then by (\(***\)) we see that the original axioms (\(*\)) uniquely determine \(S\) and \(K\); further we have the uniqueness required to define \(\lambda x. \tau\) for any term (cf. the references cited).

To establish (\(***_{n}\)), we need some lemmas. From (\(*\)) and (\(***\)) and the definitions, we can easily prove:

\[ S(u) = B^{2}(I)(S(u)) \]
\[ S(u)(v) = B(I)(S(u)(v)) \]
\[ B(u) = S(K(u)). \]

We then establish for \(n \geq 1\):

\[ B(I)(B^{n}(I)(f)) = B^{n}(I)(f), \]

because \(B^{n}(I)(f)\) has the form \(S(u)(v)\). Suppose then that, e.g.

\[ \forall x, y. z. f(x)(y)(z) = g(x)(y)(z). \]

By (\(**\)) we find:

\[ \forall x, y. B(I)(f)(x)(y) = B(I)(g(x)(y)). \]

This can be rewritten as

\[ \forall x, y. B^{2}(I)(f(x))(y) = B^{2}(I)(g(x))(y). \]

But again by (\(**\)) we find:

\[ \forall x. B(I)(B^{2}(I)(f(x))) = B(I)(B^{2}(I)(f(x))). \]

By the lemma, drop the \(B(I)\). Throw on another \(B\), use (\(**\)), drop off the \(B(I)\), and get:

\[ B^{3}(I)(f) = B^{3}(I)(g). \]

The method is perfectly general and proves (\(***_{n}\)).

The import of this axiomatization is that \(B(I)\) is the retraction of the universe \(U\) onto \((U \rightarrow U)\) and \(B^{n}(I)\) retracts onto

\[ (U \rightarrow (U \rightarrow \cdots (U \rightarrow U) \cdots)) \]

\(n\) times

We need (\(***\)) to show, e.g.:

\[ S : U \rightarrow (U \rightarrow (U \rightarrow U)). \]

We need (\(**\)) to show that the maps in these function spaces are uniquely determined by their values.

We have just been speaking in terms of models; but the calculations just carried out were formal. The axiomatic question, then, is: what is the relationship between the equational theories and the first-order theories? We shall now see the relation is a close one - even if the logic is allowed to go beyond the first order.

4. A RÔLE FOR INTUITIONISTIC LOGIC.

The (rather cheap) method of adjoining indeterminates proves that every typed or untyped theory of \(\lambda\)-calculus has an extensional model. This can also be put as a conservative extension result for theories: a \(\lambda\)-theory is an equational theory, and every such equational theory can be expanded to a first-order
theory without forcing any new equations on us. In the untyped case, the style of first-order theory is that of axioms (α), (αα), (ααα) of the previous section. These are the "logical" axioms (i.e. common to all such theories); the non-logical axioms would be all the equations between closed λ-terms demanded by whatever equational theory we started with - and these special equations could involve special "non-logical" constants.

In the typed case, we would get a many-sorted theory with a sort for each domain in the given category. As we have already pointed out, an untyped theory can always be reformulated as a typed theory by the method of retracts. So we now concentrate on typed theories - that is to say, cartesian closed categories.

But the writing down of first-order theories is not all that interesting: we clearly have nice axioms for a theory of functions, but first-order theories do not impress us as being very categorical. Such theories do not really capture the idea of the "arbitrary" function. We began our discussion with set theory, where the intention was that function spaces did really contain "all" functions - they did not just appear as an "algebra" of functions. Leaving aside for the moment the (philosophical) question of whether the desire for the ALL is a rational one, we can ask the (formal) question of whether there is a conservative extension result for higher-order theories. Surprisingly, category theorists have known the answer for some time.

Now we cannot hope to embed the theory of a typed λ-calculus into a classical higher-order theory with a full comprehension axiom of the form

\[ \forall x : A \exists ! y : B. \varphi(x,y) \rightarrow \exists f : A \rightarrow B \forall x : A. \varphi(x,f(x)). \]

Because in higher-order logic we can prove Cantor's Theorem which implies that the only type U which has a surjective map \( j : U \rightarrow (U \rightarrow U) \) is the one-element type. Thus, if a typed theory had such a type (and we know many), then the adding of the standard higher-order axioms (where we construe \( A \rightarrow B \) as the total function space of all functions) would not at all be conservative. Something else has to be tried, and the answer is higher-order intuitionistic logic.

As we shall now have to consider more than one category, let me call our given c.c.c. the category \( C \). To fix ideas, the constructions to be carried out will be done in ordinary set theory - with classical logic! The models obtained, however, will only satisfy intuitionistic logic. The obvious lack of harmony can be repaired, but it would take too much explanation here. Moreover, we are also going to assume that the given category \( C \) is a set. This is not much of a restriction, since we were thinking of \( C \) as a theory and usually a theory has a limited number of symbols in any case.

Before saying where the intuitionistic logic comes from, let me give the construction. Let \( S \) be the category of all sets and arbitrary functions; we know it is a c.c.c. The construction we need here is the well-known one of the functor category \( S^{\text{op}} \) of all contravariant functors from \( C \) into \( S \) with the natural transformations as the maps - full definitions follow.

The result is that the functor category is a model for higher-order intuitionistic logic, in particular it is a c.c.c.; moreover the original category \( C \) has a full and faithful embedding in \( S^{\text{op}} \), and this shows the conservative extension property. So much for the outline of the method, now for the details. Needless to say this represents a very early chapter in topos theory; it should be more widely known.

What is a contravariant functor? It is a mapping \( F : C \rightarrow S \) that associates to every domain \( A \) of \( C \) a set \( F(A) \) of \( S \) and to every map \( f : B \rightarrow A \) of \( C \) a function \( F(f) : F(A) \rightarrow F(B) \) (and note the change of order!) so that:
ways to calculate it owing to the associativity of composition (in \( C \)); that is why the necessary diagram commutes.

Not only are the \( H \) pleasant functors with cooperative natural transformations between them, but the by now classic Yoneda Lemma proves for us that the only natural transformations \( \eta : H_C \to H_D \) are those of the form \( H_g \) for some \( g : C \to D \). If we remark that if \( k : D \to E \), then

\[
H_k \circ g = H_{kg},
\]

this shows us that \( H : C \to S^{\text{op}} \) is a (covariant) functor between these categories. Of course \( H \) uniquely determines \( C \) and \( \eta \) determines \( g \), so we conclude from this and the Yoneda Lemma that \( H \) is a full and faithful embedding of \( C \) into the functor category (all of this on p.2 of Johnstone (1977)).

All of this discussion is "abstract nonsense" in the sense that its validity is perfectly general for any category \( C \). If we assume that \( C \) is a c.c.c., then we can say more. The point is that \( S^{\text{op}} \) is a very powerful category. For example, it is always a c.c.c. even if \( C \) is not. The cartesian closed structure of the functor category is obtained through the following definitions.

Before getting down to details, however, some more vivid terminology might help. Think of a functor \( U \) in \( S^{\text{op}} \), following Lawvere, as a "variable domain". That is to say, for each \( A \in C \) we have an associated domain (set) \( U_A = U(A) \). The maps \( f : B \to A \) in \( C \) give us transitions between "stages" \( A \) and "latter" stages \( B \); and each such transition "restricts" elements in \( U_A \) to elements in \( U_B \) "along" the map \( f \). To save writing, let us set \( af = (Uf)(a) \) when the functor \( U \) is understood. That \( U \) is indeed a functor comes down to these equations:

\[
(af)g = a(f \circ g),
\]

This means \( v_B \circ F(f) = G(f) \circ v_A \). An example will help explain this.

For each \( C \) of \( C \), let

\[
H_C(A) = \{ h \mid h : A \to C \}
\]

and if \( f : B \to A \in C \), let \( H_C(f) \) be the map taking \( h \in H_C(A) \) into \( h \circ f \in H_C(B) \). It is easy to show \( H_C \) is a (contravariant) functor. It is often called the representable functor (corresponding to \( C \)), and we shall see that it is very "representative".

Now let \( g : C \to D \) in \( C \). There is a natural transformation \( H : H_C \to H_D \); because, for each \( h \in H_C(A) \) we can map it to \( g \circ h \in H_D(A) \), naturally. The composite map for \( f : B \to A \) takes \( h \in H_C(A) \) into \( g \circ h \circ f \in H_D(B) \), and there are two equal
where \( f : B \to A \) and \( g : C \to B \) in \( C \). To define a functor \( U \), then, we must just give the domains and the restrictions. For example the unit functor \( I \) has \( I_A = \{0\} \), a one-point set, and all restrictions constant \( 0 \) if \( f = 0 \). And all natural transformations into \( I \) are constant.

Now suppose \( U \) and \( V \) are two functors. We define \( U \times V \) so that for all \( A \) in \( C \)

\[
(U \times V)_A = U_A \times V_A
\]

and whenever \( a \in U_A \) and \( b \in U_A \) and \( f : B \to A \), then

\[
(a, b)f = (a1f, b1f).
\]

(Note that the restriction symbol above is used in three different senses.) The natural transformations \( p : U \times V \to U \) and \( \eta : U \times V \to V \) have obvious pointwise definitions (e.g.,

\[
p_A = p_{U \times V} : (U \times V)_A \to U_A
\]

and they clearly commute with restrictions. Similarly, if \( \mu : W \to U \) and \( \nu : W \to V \) are given natural transformations, then \( \langle \mu, \nu \rangle : W \to U \times V \) is also defined pointwise:

\[
\langle \mu, \nu \rangle_A = \langle \mu_A, \nu_A \rangle : W_A \to U_A \times V_A.
\]

Again it is obvious that these maps commute with restrictions, so \( \langle \mu, \nu \rangle \) is natural. As all of this is pointwise, the verification of such equations as \( p \circ \langle \mu, \nu \rangle = \mu \) is easy.

Again suppose \( U \) and \( V \) are given. In defining \( (U \to V) \), we cannot be quite as pointwise. That is, \( (U \to V)_A \) cannot be taken simply as the set of functions \( U_A \to V_A \), the function space in sets. The reason, roughly, is that when we have a function at one stage, we also have to know how it restricts at later stages; a simple mapping from \( U_A \) into \( V_A \) does not give us enough information for that. When \( f : B \to A \), restriction on \( U_A \) maps into \( U_B \); this is the wrong direction for us to be able to pass from an arbitrary function defined on \( U_A \) to one defined on \( U_B \). So an element of \( (U \to V)_A \) has to be a whole family of functions

\[
\varphi_f : U_B \to V_B,
\]

one for each \( f : B \to A \). (Note: \( A \) is fixed, \( f \) and \( B \) are variable.) Moreover, we must assume that all is harmonious with restrictions: \( c \circ g = \varphi_{f \circ g} \) (b1g) whenever \( c = \varphi_f(b) \), for \( b, c \in U_B \) and \( g : C \to B \) in \( C \). In words, \( \varphi_A \) is the "present" function; while \( \varphi_f \) is what becomes of it in the "future", supposing time evolves along \( f \). Now families \( \varphi \) of this kind in \( (U \to V)_A \) have to be restricted. By what we just said, the following is more or less forced upon us:

\[
(\varphi1f)_g = \varphi_{f \circ g}
\]

where \( f : B \to A \) and \( g : C \to B \), so that \( \varphi1f \) is a family in \( (U \to V)_B \).

That defines \( (U \to V) \) as a functor. To have \( S^{\text{op}} \)

\[\text{be a c.c.c., certain maps } \varepsilon \text{ and } \Lambda \text{ are, alas, still required. The evaluation map } \varepsilon : (V \to W) \times V \to W \text{ is fortunately rather clear (as a natural transformation). Suppose } \varphi \in (V \to W)_A \text{ and } a \in V_A. \text{ Then}
\]

\[
\varepsilon_A(\varphi, a) = \varphi_A(a),
\]

so \( \varepsilon_A : ((V \to W)_A \times V_A) \to V_A. \text{ If } f : B \to A, \text{ then}
\]

\[
\varepsilon_A(\varphi, a) \circ f = \varphi_A(a \circ f) = \varphi_f(a1f)
\]

\[
= (\varphi1f)_B(a1f) = \varepsilon_B(\varphi1f, a1f)
\]

This proves that \( \varepsilon \) is natural.

Next, suppose \( \psi : U \times V \to W \) is natural. Define

\( \Lambda \psi : U \to (V \to W) \) by
where for \( a \in U_A, b \in V_B, \) and \( f : B \to A \) we have:

\[
(\langle \psi \rangle_A)_f(b) = \psi_B(a \upharpoonright f, b).
\]

To show \( \psi \) is natural, we must calculate:

\[
(\langle \psi \rangle_A f)_g(c) = (\langle \psi \rangle_A)_f(g \circ c) = \psi_B(a \upharpoonright f \circ g, c) = (\langle \psi \rangle_B)_g(a \upharpoonright f(c)
\]

for \( a \in U_A, f : B \to A, g : C \to B, c \in U_C. \) It follows that

\[
(\langle \psi \rangle_A)_f = (\langle \psi \rangle_B)_g(a \upharpoonright f).
\]

We have to leave to the reader the verification of the two basic equations of c.c.c.'s involving \( \epsilon, A, p \) and \( q. \) As there was only one way that the definitions could be written, the verification is quite mechanical, however.

As I said before, the functor category is "powerful", and indeed it is much more than a c.c.c. For instance, we can define the analogue of the power set for arbitrary functors. For any \( U, \) let \( (PU)_A \) be the collection of all families \( S_f \) indexed by \( f : B \to A \) where \( S_f \subseteq U_B \) such that \( b \uparrow g \in S_f \) whenever \( b \in S_f \) and \( g : C \to B \) in \( C. \) Restriction is defined by

\[
(S \upharpoonright f)_g = S_{f \circ g}.
\]

The significance of the power operator will become clear when we speak about higher-order logic.

Having seen why the functor category is a c.c.c., it is good to pause a moment to appreciate the difference between the elements \( a \in U_A \) as sets and the "elements" of \( U \) in the categorical sense. If \( a : I \to U \) is natural, it means that \( a : a : I_A \to U_A \) in \( S. \) Let \( a_A = a_A(0), \) then \( a_A \in U_A. \) If \( f : B \to A, \) then because \( a \) is natural we find:

\[
a_A \uphrasemid f = a_A(0) \uphrasemid f
\]

\[
= a_B(0 \upharpoonright f)
\]

\[
= a_B(0)
\]

\[
= a_B.
\]

This is very strong indeed, since usually if \( a \in U_A \) and \( f_0, f_1 : B \to A, \) there is no reason why \( a \upharpoonright f_0 = a \upharpoonright f_1. \) So the number of "elements" of \( U \) will very likely be rather small.

(And, even worse, \( a \) has to be chosen for all \( A \) in \( C. \))

In the special case \( \sigma : I \to PU \) we can simplify the choices out of \( (PU)_A \) even further. Write \( S_A = \sigma_A(0)_A, \) then \( S_A \subseteq U_A \) for all \( A \) in \( C. \) Moreover, when \( f : B \to A, \) then

\[
S_B = \sigma_B(0)_B
\]

\[
= \sigma_B(0 \upharpoonright f)_B
\]

\[
= (\sigma_A(0) \upharpoonright f)_B
\]

\[
= \sigma_A(0)_B
\]

This means that \( \sigma_A(0) \) is determined from the \( S_B \)'s. And if they are chosen so that

\[
b \in S_B \text{ implies } b \upharpoonright g \in S_C
\]

whenever \( g : C \to B, \) then the \( \sigma_A \) so defined from them provides a natural transformation. Again, we see the elements of \( PU \) are rather special. We can say that elements of a functor provide information about the "global" nature of the functor; but this is far from determining it, for there can be considerable "local" activity that cannot be sensed globally. For example, the sets \( U_B \) can be empty for a long "time", only becoming non-empty in the "future". The functor \( U \) is not trivial, but it has no global elements.
We should also pause to see why the functor $H$ maps $C$ into a subcartesian closed category of $S^C$ (up to isomorphism). It is easy to check that the functors $H_A \times H_B$ and $H_{A \times B}$ are naturally isomorphic. We also have to do the same for $(H_A \to H_B)$ and $H_{A \to B}$. Consider an element of $(H_A \to H_B)_C$. It is a family of maps

$$\varphi_f : H_A(D) \to H_B(D),$$

for $f : D \to C$. In particular consider the standard maps

$$p : (C \times A) \to C$$

and $q : (C \times A) \to A$. Then $\varphi_p(q) : (C \times A) \to B$. So, since $C$ is a c.c.c., we find $\varphi_p(q) \in (H_A)_C$. In the other direction, let $t : C \to (A \to B)$. Define $\tau_f$ for $f : D \to C$ by

$$(\tau_f)(g) = \epsilon \circ \langle t \circ f, g \rangle$$

where $g : D \to A$. We see this lies in $H_B(D)$. Now

$$\epsilon \circ \langle t \circ f, g \rangle \circ k = \epsilon \circ \langle t \circ f \circ k, g \circ k \rangle$$

whenever $k : E \to D$. Thus,

$$(\tau_f)(g) \circ k = \tau_{f \circ k}(g \circ k).$$

This proves that the family $\tau_f$ lies in $(H_A \to H_B)_C$. It has to be left without proof that these two correspondences are inverse to one another and provide a natural isomorphism.

Well, this is a rather heavy construction starting from one little category $C$. The question: what does it prove? Why worry about the functor category? The answer is that the functors give an interpretation of higher-order logic, as we hinted earlier, and now we have to pay up and demonstrate how to construct logical formulae. The idea from topos theory when specialized to the functor category looks very much like Kripke models of intuitionistic logic – except that the "times" form a category $C$ rather than just a partially ordered set, as has often been emphasized by Lawvere (see, e.g. Lawvere (1975)).

To make the logical language more definite, let us think of the domains $A$ in $C$ as being (in a one-one correspondence with) type symbols. Introduce new type symbols built from the ones in $C$ (the "ground" types) by forming $\exists F$, $(T \times S)$, $(T \to S)$, $PT$ for all type symbols. (Note: $A \times B$ in $C$ is being distinguished from the type symbol $A \times B$. But the "meaning" of the symbol $A \times B$ will turn out to be something "isomorphic" to $A \times B$ in $C$. The trouble is that the domain $A \times B$ does not in itself determine the $A$ and the $B$; whereas the type symbol does.) We extend the notation $H_A$ to $H_T$ for any type symbol in the obvious way; that is, $H_T \times S$ is the product $H_T \times H_S$ in the functor category. This is the first step in treating the functor category as an interpretation of a higher-order theory.

Next we must imagine a logical language with a supply of variables of each type. Atomic formulae will be of these forms: $i ; x = y$, where $x$ and $y$ have the same type; $y = Fx$ where $F$ is a constant symbol corresponding to a map $f : A \to L$ in $C$ and $x$ has type $A$ and $y$ type $B$; $z = (x,y)$ where $x$ has type $T$, $y$ type $S$, $z$ type $T \times S$; $z = \lambda(x,y)$, where $z$ has type $S$, $y$ has type $T$, and $x$ type $(T \to S)$; $y \in x$, where $y$ has type $T$ and $x$ type $PT$. Atomic formulae are then made into compound formulae by the usual constructs: $\phi \land \exists \phi, \phi \lor \exists \phi, \phi \to \exists \phi, \forall x. \phi, \exists x. \phi$.

Suppose $A$ is a domain of $C$, $\phi$ is a formula, and $s$ is a valuation of the free variables of $\phi$. We are going to define what Joyal-Reyes (1980) call the forcing-satisfaction relation $AH$ of $[s]$. The definition here will be in one respect simpler than theirs since the category $C$ carries no topology; in another respect it is more complicated because we have the whole higher-order language. But the adaptation is straightforward. Before we can give the clauses, we must say what kind of a creature $s$ is. We must make $s$ relative to $A$ in the first place. So if $x$ has type $T$, then $s(x)$ is to belong to the set $H_T(A)$. Now here are the clauses:
We only have to take care that we remember that some ranges of variables can be empty (that a set \( H_T(A) \) may be empty), and so the logic is the so-called "free" logic (cf. Scott 1979) for a discussion.

In order to verify the special axioms of higher-order logic, we need to remark first on what Joyal-Reyes call the "functorial" character of \( \vdash \):

\[
\text{if } A \vdash \Phi[s] \text{ and } f : B \to A, \text{ then } B \vdash \Phi[s \upharpoonright f].
\]

This, too, is a property familiar from Kripke models. It plays a direct role in the verification of the comprehension axiom:

\[
\forall u_0, \ldots, u_{n-1} \exists x \forall y \left( y \in x \leftrightarrow \Phi \right)
\]

where the free variables of \( \Phi \) are among \( u_0, \ldots, u_{n-1}, y \) and \( x \) is a new variable not free in \( \Phi \) of type \( \mathcal{P}T \) where \( T \) is the type of \( y \).

To show the above valid in the interpretation we only have to show that for every \( A \) of \( C \) and for all \( b_0, \ldots, b_{n-1} \) in the \( H_S(A) \) of the appropriate types \( S \), there is an element \( c \in H_{\mathcal{P}(T)}(A) \) such that

\[
A \vdash \forall y \left( y \in x \leftrightarrow \Phi[y] \right).
\]

Here \( s \) is the valuation where \( s(x) = c \) and \( s(u_i) = b_i \). We have to define \( c \). For each \( f : B \to A \), let

\[
c_f = \left\{ t \in H_T(B) \mid B \vdash \Phi[s \upharpoonright f(t/y)] \right\}
\]

The functorial character of \( \vdash \) proves for us that \( c \in H_{\mathcal{P}(T)}(A) \). It is now easy to check from the clauses of the definition of \( \vdash \) that at \( A \) the above formula is indeed forced.

In a similar way we can verify the functional version of comprehension:

\[
\forall x \exists y \forall z \left( z = y \leftrightarrow \Phi \right) \to \exists f \forall x \left( z = f(x) \leftrightarrow \Phi \right),
\]
where we have \( x \) of type \( T \), \( y \) and \( z \) of type \( S \); \( f \) of type \((T \to S)\) and \( y \) not free in \( \Phi \). Again, the functorial character of \( \vdash \) connects with the way we had to define \( (H_T \to H_S) \).

We also have to verify such extensionality properties as:
\[
\forall f, g (\forall x, y [y = f(x) \iff y = g(x)] \to f = g),
\]
\[
\forall x, y (\forall z [z \in x \iff z \in y] \to x = y),
\]
where the variables have to be given the appropriate types.

But in defining the function spaces and the powersets in the functor category, we only put in just enough of a mapping or a set to get an appropriate functorial character. Hence, if two such objects are extensionally equal by the formulae above, they will be equal. This has to be spelled out via \( \vdash \), but it is not surprising.

The higher-order axioms for ordered pairs are obvious, and their satisfaction relates at once to the definition of product of functors. As for the embedding of \( C \) into the higher-order theory we find
\[
\forall x, y (y = fx \iff y = gx)
\]
is valid if and only if \( f = g \) in \( C \). Also, when \( h = g \circ f \), we have as valid
\[
\forall x, y, z ([y = fx \land z = gy] \to z = hx).
\]
Further, functions like \( f \) are well defined:
\[
\forall x \exists y \cdot y = fx
\]

There are many principles of identity that should be mentioned, but we will not write them down here. Among them we would also find the statements that there is a unique element of type \( \mathbb{1} \) and all maps of type \((T \to \mathbb{1})\) are constant. (Perhaps the constant 0 should figure in the language, but it is not all that essential.)

As for questions of uniqueness, if the sentence
\[
\forall x \exists y \forall z (z = y \iff \Phi)
\]
is valid (i.e. forced at all \( A \)), then provided \( x \) and \( y \) have ground types \( B \) and \( C \), respectively, there is an \( f : B \to C \) in \( C \) such that
\[
\forall x, z (z = fx \iff \Phi)
\]
is valid too. The validities, then give us an exact picture of \( C \) at the level of ground types: the higher-order theory is conservative over \( C \).

But the higher-order intuitionistic theory of the functor category is much more than just a conservative extension; it is a full-blown higher-order theory with full comprehension axioms. That is to say, we started out with a category \( C \) we regarded algebraically as a theory of functions. Well, the construction of the functor category shows us that we can indeed construe \( C \) and its maps as normal, everyday functions in a normal, every-day higher-order logic. This works as long as we agree to keep our logic intuitionistic. But experience with intuitionistic logic really shows that the system is a natural one and that it leads to very, very interesting theories. Even if \( C \) is a c.c.c., we can show that the embedding of \( C \) in the higher-order logic preserves all the cartesian closed structure, so that the function spaces in \( C \) really become spaces of all possible functions in the higher-order theory. The principles of \( \lambda \)-calculus are thus consequences of the standard logical axioms. This seems to me to establish complete harmony between (intuitionistic) logic and (typed) \( \lambda \)-calculus.

The next step in this investigation would be to see what other properties of the higher-order logic could be enforced and still preserve the conservative extension over the given category \( C \). The functor category is just a very first stage of the investigation: in topos theory the categories of sheaves result
from putting a kind of modal operator into the logic, and making a reinterpretation of the logical connectives and quantifiers.

The passage from $S^{\text{op}}_{\text{c.c.c.}}$ is one of finding c.c.c. as cartesian closed subcategories of the functor category. There are many of them and many still contain $C$ as a cartesian closed subcategory. So, there is much to look for, and – I am sure – much left to be discovered of definite logical interest.

5. TYPE-FREE DOMAINS REVISITED

Having made any given c.c.c. $C$ "honest" as a theory of functions in higher-order logic, we can conclude from the method of retracts of Section 3 that any type-free $\lambda$-theory can similarly be made honest. Intuitionistic logic is very tolerant of types $U$ where $(U \to U)$ is a retract; so tolerant in fact that any $\lambda$-theory can be embedded in a suitable higher-order logic. Self-application is no longer odd: it is something that may very well turn up when we weaken our logic to be intuitionistic but still require that functions spaces like $(U \to U)$ contain all functions.

This provides a certain kind of rescue for the type-free calculus, but the move fails to give it a universal role: the creators of the type-free theory hoped that such a universe $U$ could be thought of as containing all the functions there were. We shall not try to go so far in the present context, but various constructions can be used to show that not only is it possible to have one such type-free domain, but it is always possible to find them being richer and richer and containing more and more functions. Only a sketch of the construction can be given here.

Suppose, for the sake of illustration, we have some types $A$, $B$, $C$ that we happen to like, and that we are interested in the functions between them — possibly also in functions of the type $(A^2 \times B) \to C$, and similar multivariable types. We could probably work up a c.c.c. containing $A$, $B$, $C$ and these functions, but the straight-forward construction would contain no type-free domains (cf. the category of sets and maps in ordinary logic). We need a new method. My first approach is to use the idea of continuous lattices. I do not want to go into a lot of detail (cf. Gierz et al. (1980) for just such details), but there is an easy definition that can be invoked at least to make the statements precise.

We shall employ what are not technically lattices but "half" lattices without unit elements (top elements). Fortunately we do not have to go into a long list of definitions, since I have been able to characterize them neatly as special topological spaces. They are in fact $T_0$-spaces (i.e. spaces where points are uniquely determined by their neighborhoods) $D$ such that whenever $X$ is a dense subspace of a topological space $Y$, and $f : X \to D$ is a continuous function, then $f$ has a continuous extension $\tilde{f} : Y \to D$. What I proved is that the category of such spaces $D$ together with continuous maps between them is a c.c.c. (There are very many intriguing c.c.c.'s related to the category of topological spaces!) Let us employ the temporary name "injective" for these spaces.

As an example of injective spaces, consider one of our given types $A$, which for simplicity we construe just as a set.
The injective space $A_\bullet$ corresponding to $A$ results from adding one new point $\bullet$. Or, if classical logic is not assumed, we take $A_\bullet$ as the space of subsets of $A$ with at most one element.
The topology is generated by sets of the form $\{x \in A_\bullet \mid a \in x\}$ where $a \in A$. Thus, a function $f : X \to A_\bullet$ is continuous iff $\{x \in X \mid a \in f(x)\}$ is open in $X$ for each $a \in A$. Now if $X \subseteq \tilde{Y}$ as a dense subspace, we have only to define

$$\tilde{f}(y) = \bigcup \{\{a \in A \mid \forall x \in N \cap X. a \in f(x)\} \mid y \in N\},$$

where $N$ ranges over the open sets of $Y$. Because every non-empty open set has a non-empty intersection with $X$, it follows
that \( \bar{T} : Y \to A_k \). To prove \( \bar{T} \) continuous, we remark that
\( \{ y \in Y \mid a \in \bar{T}(y) \} \) is the largest open subset of \( Y \) whose inter-
section with \( X \) gives \( \{ x \in X \mid a \in f(x) \} \). It is also easy to
calculate that \( \bar{T} \) extends \( f \). So \( A_k \) is injective. Note, too,
that \( A \) may be regarded as a dense subspace of \( A_k \) if we map \( a \) to \( \{ a \} \). Hence, every function \( g : A \to B \) has a unique counterpart
\( g_k : A_k \to B_k \) so that the "restriction" of \( g_k \) to \( A \) gives \( g \) back
again (indeed \( g_k(\{ a \}) = \{ g(a) \} \)). This really means that the
\( * \)-construction is a faithful functor from the category of our
sets \( A, B, C \) into the category of injective spaces and contin-
uous functions.

But now we can apply my construction of \( \lambda \)-calculus models to
find an injective space \( U \) which, in the category of injective
spaces, has \( (U \to U) \) as a (continuous) retract and in addition
has \( A_k, B_k, C_k \) as retracts. (In fact, for those who know
the method, we solve the domain equation
\[ U = A_k \times B_k \times C_k \times (U \to U), \]
where of course a factor is always a retract of a product; be-
because in the category of injective spaces the one--point space
is a retract of every space.) This idea could be extended to
obtain any given set of injective spaces as retracts of a
single space \( U \).

Next we invoke the plan of the previous section using as the
category \( C \) the retracts of \( U \) (and continuous functions), which
we can regard as a small category (as a set). The functor cat-
geropy has all higher-order logic as well as a full and faith-
ful picture of \( C \). We are definitely going to take advantage
of the higher-order structure in looking at subtypes of the
functors \( H_y \) where \( V \) is a domain in \( C \) -- more precisely, we will
look at the category generated by certain of these subtypes or
subfunctors in the functor category.

In the first place, consider \( H_{\bar{A}_k} \). Let \( K_{\bar{A}_k} \) be the functor
where \( K_{\bar{A}_k}(V) \) is just the set of all continuous functions
\( f : V \to A_k \) where for some \( a \in A_k \), we have \( \{ a \} \in f(x) \) for all \( x \in V \).
(The retracts of \( U \) are simply being regarded as injective
spaces, and we do not distinguish between \( A_k \) as a constructed
space based on the set \( A \) and as a retract of \( U \).) The restric-
tion operation \( 1_f : K_{\bar{A}_k}(V) \to K_{\bar{A}_k}(W) \) is the one for the functor
\( H_{\bar{A}_k} \) with the domain cut down: \( K_{\bar{A}_k} \) is a subfunctor of \( H_{\bar{A}_k} \). We
can think of the maps in \( K_{\bar{A}_k}(V) \) as being the constant maps with
values in \( A \). So what then can we have for natural transforma-
tions \( \nu : K_{\bar{A}_k} \to K_{\bar{B}} \), the maps in the functor category? Well,
imagine one. Now if \( a \in A_k \), we can take the appropriate con-
stant map \( k_{\bar{A}_k} \in K_{\bar{A}_k}(\{ a \}) \), where \( \{ a \} \) is the one--element space. Then
\( \nu(k_{\bar{A}_k}) \in K_{\bar{B}}(\{ a \}) \). But this too is a constant map and determines a
unique \( b \in B \); so \( \nu \) defines a function \( \nu : A \to B \). And, since
every constant map factors through the one--element space \( \{ a \} \), the
map \( \nu \) uniquely determines \( \nu \). But any map from \( A \) into \( B \) can
be made to turn up in this way by trivially fooling around with
constants. We conclude, therefore, that \( K_{\bar{A}_k} \) as a functor -- of
our given sets \( A -- \) is a full and faithful embedding.

What have we done? First, starting in sets -- or perhaps,
better, in higher-order logic -- we found (or gave ourselves)
a category of types we liked. To be more definite, they could
have been unioned together so they were all subtypes (subsets)
of a single set, \( V \), say. We then embedded faithfully this
category of sets and maps into the category of injective spaces
via the very elementary \( A_k \) construction. Of course, the spaces
\( A_k \) are very special, so the "universal" space \( U \) with \( (U \to U) \) as
a retract is much more messy than \( V_{\bar{A}} \). But the category of re-
tracts of \( U \) contains all the maps between the \( A_k \). Finally this
category of retracts is fully and faithfully embedded in the
the functor category. The latter has the advantage of subtypes in profusion, so we were able to recapture the original category of subsets of \( V \) as a full and faithful subcategory of the functor category.

And having done all this, what have we bought? Well, the (pictures of) the \( A \)'s were subtypes of the \( A_A \)'s which are both retracts and subtypes of \( U \), and in the functor category \( U \) is a model for the untyped \( \lambda \)-calculus. So that means that starting with our original notion of function, we have - in the logic of the functor category - consistently been able to assume that there are types giving models for the "type-free" \( \lambda \)-calculus, and further, that these types are rich enough to contain our original category in a full and faithful way. In more detail: the new logic allows us to think of \( A \) and \( B \) as subtypes of \( U \), where \( (U \rightarrow U) \) is a retract, so that any function from \( A \) into \( B \) is the result of restricting a function in \( (U \rightarrow U) \) down to the subset \( A \). Warning: this does not hold for all subtypes of \( U \), the \( A \), \( B \), \( C \) were given in advance and \( U \) was constructed relative to them. Still, this means that even in models for type-free \( \lambda \)-calculus (which can be regarded as ordinary function spaces), we are not losing sight of the standard idea of function. To have \( (U \rightarrow U) \) as a retract of \( U \), the functions have to "bend" a little, but we have kept them "straight" as far as the given \( A \), \( B \), and \( C \) go.

We have just shown how a type-free domain \( U \) can incorporate given domains as well as the "arbitrary" functions on them. In Engeler (1979) it is shown that \( \lambda \)-calculus models can also incorporate any algebra; specifically it is shown that any algebra can be made isomorphic to a subset of the model where the operation is functional application itself. We shall give a proof here using the constructs we have mentioned.

Let \( A \) be set. An "algebra" can be regarded as any binary operation \( \cdot : A \times A \rightarrow A \). A partial algebra can be taken to be any continuous \( \cdot : A_A \times A_A \rightarrow A_A \). It is easy to argue that any algebra on \( A \) determines a partial algebra on \( A_A \).

Now let the \( \lambda \)-calculus model \( U \) be taken so that \( U = A_A \times (U \rightarrow U) \). We regard elements \( x \in U \) as pairs \((x_0, x_1)\). The application operation \( f(x) \) on \( U \) can be defined as \( f_1(x) \) because \( f_1 \in (U \rightarrow U) \). Now recall that \( \lambda \)-calculus models satisfy the fixed-point theorem; so we can define a map \( \rho : A_A \rightarrow U \) by the functional equation:

\[
\rho(a) = (a, \lambda x : U \cdot \rho(a \cdot x_0)),
\]

where the \( \lambda \)-operator gives an element in \( (U \rightarrow U) \). This map is continuous and one-one into. Now calculate in \( U \):

\[
\rho(a)(\rho(b)) = \rho(a \cdot \rho(b)_0)
= \rho(a \cdot b)
\]

So the image of \( \rho \) is closed under application, and the resulting applicative subalgebra is isomorphic to the given algebra \( \langle A_A, \cdot \rangle \).

This result would seem to have a potentially useful implication for non-extensional models of combinatorial algebra: any such can be embedded in an application-preserving way into an extensional model. This works even if we regard application as a partial operation. Warning: we do not obtain a combinator-preserving embedding, however. That is, if the algebra \( \langle A_A, \cdot \rangle \) has elements \( S \) and \( K \), satisfying the usual equations in \( A_A \), we cannot conclude that the embedding \( \rho : A_A \rightarrow U \) will map the \( S \) and \( K \) of \( A_A \) to the "true" \( S \) and \( K \) of the \( \lambda \)-calculus model \( U \). The "functions" in \( A_A \) operate only on \( A_A \), which is quite a limited part of \( U \); clearly \( \rho \) does not give elements \( \rho(a) \in U \) very broad roles. But at least we can say that anything that even looks a little like application can be assumed to be application in a suitable domain.
great philosophical interest since it does not relate the idea
of λ-calculus to any broad notion of functions. This desire
was taken care of by:

4. Every c.c.c. can be fully and faithfully embedded in an
intuitionistic theory of types with the full (impredicative)
power-set construct and function spaces (higher-order intu-
itionistic logic).

The domains of the c.c.c. become types in the theory. The
word "fully" means that the definable maps between the types
all come from maps in the category; "faithfully" means that in
the higher-order theory no new equations between these maps are
introduced over what we already had in the category. In other
words, this is a conservative extension result. It has been
known for quite a time in category theory, and the functor
category we employed in the construction is one of the
very first examples of a topos; there must be considerable use
possible of more interesting examples of topos.

However, there was already enough philosophical interest in
this easy construction. Namely, it was seen that equational
λ-calculus is perfectly consistent with higher-order logic
where - provided we only employ intuitionistic logic - we can
speak of function spaces in the normal way in type theory.
Some people can, if they like, stick to λ-terms and equations;
but others can use whatever logical means they like for dis-
cussing functions. However, if the logician proves in his
higher-order theory that a certain property picks out a func-
tion \( f : A \to B \), then, if \( A \) and \( B \) are from our given category,
this definable \( f \) must be given by a standard λ-term. So the
logic in that sense gives nothing new, but at least we know
that the sense for λ-calculus is exactly that it can always be
taken to be talking about functions and full function spaces in
a higher-order theory.
Turning things around the other way, it is interesting to see from what we know about "type-free" theories that intuitionistic logic allows for reflexive domains — and even lots of them. It would be even more interesting if it were possible to strengthen higher-order logic (say, by adding some new primitives), so that we could express in the logic the axiom that every c.c.c. (a structure satisfying a simple first-order statement) had a full and faithful representation in a category of subsets (better: quotients of subsets) of a reflexive domain. I conjecture that this is possible and that the theory can be taken conservative over any given c.c.c.

Though we did not prove such a sweeping result, we did sketch a proof of:

5. Every given c.c.c. can be realized fully and faithfully as a category of subtypes of a reflexive type in a higher-order theory.

We did not work out the cartesian closed details of this assertion but contented ourselves with showing how to accommodate a finite number of types. By the way, it should be remarked that 4 and 5 hold for an arbitrary (small) category, so in particular we have the (known) result that every category can be conservatively extended to a c.c.c. The method of proof for 5 was via the author's original construction of λ-calculus models, which can be conveniently carried out in the category of "injective" $T_0$-spaces. There are many variations on this construction, and it might be interesting to see how different the different categories with reflexive domains really are from the point of view of higher-order logic.

Another remark: the construction has many connections with Curry's Theory of Functionality (i.e. the problem of finding other c.c.c.'s inside a λ-calculus model). But as I have indicated several times it is really better to work with equivalence relations (on subtypes of a reflexive domain $U$) because in typing the functions we have to make them hereditarily extensional in order to be able to have a category. Thus a reflexive domain has (at least) two interesting c.c.c.'s associated with it: the category of retracts and the category of equivalence relations. The second, by the way, contains the first as a sub-c.c.c.

Finally we recalled a result of Engeler which was very appropriate to the present discussion:

6. Any (partial) algebra can be isomorphically represented as an applicative subalgebra of a reflexive domain.

I think that this means in particular that non-extensional theories of functions can be subsumed under the extensional theory: the non-extensional function algebras are just subalgebras of normal function algebras. There is certainly a conservative extension result here, but whether it helps to prove any new theorems is another question.

What has to be investigated next, I think, is the problem of how strong the higher-order theories can be made and still have them as conservative extensions of given categories. Much is known in topos theory about constructions of categories of sheaves (these are subcategories of the functor category), but much remains to be explained to the logician. Thus, there are several interesting categories made up out of continuous functions or out of computable functions (when we look at them from the outside), but what we would like to know is what logical sentences (internal properties) are satisfied for the various functor or sheaf categories. The so-called Church's Thesis (all number-theoretic functions are recursive) or Brouwer's Theorem (all real functions are continuous) are cases in point,
and they are satisfied in certain toposi. It would be an important next step for λ-calculus to relate these model constructions to interpretations of λ-calculus. The author hopes that the present paper will encourage others to look further.

REFERENCES