D_∞, Category Theory, and All That

Ryan J. Wisnesky
Philosophy 351B

June 2006

Introduction

There are two main approaches to constructing and investigating models of the untyped lambda calculus (λ-models). One approach, taken in [1], is to present the models in familiar, set-theoretic terms. This approach has the advantage that the concepts involved are intuitive and that the models, in this setting, feel very much like we would expect λ-models to feel.

The other approach, which has arguably become more popular over the years, is to present λ-models in a category-theoretic setting. This approach has the advantage that once the requisite category theory is mastered, the definition of a λ-model and the machinery required to investigate them become much simpler. Unfortunately, category theory, even though it is becoming the lingua franca of many sub-disciplines of computer science (for instance, denotational semantics), is abstract and not as intuitive as set theory. Although there are numerous books that purport to offer "category theory for the working computer scientist" [5], these books often develop category theory in detail and only then examine its applications to computer science.

To complicate matters even further, some texts [3] freely mix a set-theoretic and category-theoretic approach. This mixing is also found in books on typed lambda calculus [4].

The goal of this paper to leverage category theory to help explain λ-models, *while simultaneously leveraging λ-models to help explain category theory.* Because this paper aims to explain two unintuitive constructs in terms of each other, this work will be most useful to the student who is already familiar with models of the typed lambda calculus and their set theoretic construction, because many of the concepts found there also apply in this setting.

The outline of this paper is as follows: first, we show that constructing λ-models is not trivial. Then, *just enough* category theory is presented to allow for further connections to category theory to be made *while we are developing λ-models.* Thus, the connections to category theory will appear "naturally" and where they are most relevant. General properties of λ-models are then discussed, and applicative structures and several types of "easy" model are introduced. Scott's topology and the model D_∞ are then presented.

The bulk of the set-theoretic material in this paper is taken from [1], while most of the category-theoretic material is taken from [4]. Various other sources are cited as appropriate. Regardless of the source, the material is re-ordered and modified to suit our needs; namely, less of an emphasis on proofs, more emphasis on intuition, and a parallel set- and category-theoretic development.

1 Why are λ-models Difficult?

The purpose of a λ-model is to associate with each λ-term an *interpretation* of that term such that provably equal terms are interpreted the same. In addition, we do not want there to be "junk" in the model; that is, elements in the model that are not the interpretation of any term. Nor do we want terms to be equated in the model without good reason. It is also desirable that the domain of the model – the domain from which interpretations are drawn – be a "semantic" domain that is close to what we believe the terms "actually are." For instance, one model of Peano arithmetic is the set of natural numbers. In this model, the constant symbol **1** is interpreted as the natural number 1.

We would like to examine what λ-models are like in general, and also to develop "natural" models that correspond with our intuition of λ-terms as functions "on themselves."

Unfortunately, it is impossible for such a straightforward model to exist. Consider the lambda term $\lambda x.x$ and suppose we have a model with domain D. Then, using $[\![M]\!]$ to stand for the interpretation of term M, we have that $[\![\lambda x.x]\!] \in D$, because it is a term, and all terms must have an interpretation in D. But notice that this term is also a function – in fact, it is the identity function – which means it must have an interpretation in $D \to D$, which is D's function space. This argument may be generalized: all λ-terms are functions, and hence their interpretations belong in $D \to D$, and all λ-terms are terms, and hence their interpretations belong in D. Therefore, it must be the case that $D \cong D \to D$. If we simply let $D \to D$ be the set of all functions from D to D, then the cardinality of $D \to D$ will be $2^{|D|}$, and so the required isomorphism cannot exist, because a set of cardinality N cannot be put in a one-to-one correspondence with a set of cardinality 2^N. The consequences of this simple observation are felt throughout the model theory of untyped lambda calculus.

2 Just Enough Category Theory

There are numerous introductions to category theory; c.f. [5] However, most of these texts present an extensive development of the theory before applications to semantics are introduced. The introduction to category theory found in [4] section 7.2 is quite accessible and relatively short, and these definitions are taken from there. It is often the case that the definitions in category theory are straightforward but different enough from other branches of mathematics to make developing an intuition about category theory difficult, and so this section stresses the intuitive meanings of the definitions.

Category theory is the study of abstract structure – abstract enough that it is a long-standing joke that category-theoretic results are said to be "true by abstract nonsense." Fundamentally, category theory seeks to understand structures and maps between structures; a category is simply a collection of *objects* and *arrows* between objects. (Arrows are sometimes called *maps* or *morphisms*.) One classic example is the category of groups, where the objects of the category are groups and the maps between groups are group homomorphisms. There is even a category of all sets: the objects are sets and the arrows are set-theoretic functions. (Of course, the domain of this category itself cannot be a set, as then the domain would be the paradoxical set of all sets.) It is often the case that the category of sets is the underlying intuitive device that we use to reason about how arbitrary categories work; however, the intuition that the objects in a category are themselves collections is not valid and can sometimes lead to trouble.

A category is an ordered tuple $\langle C^o, C^d, dom, cod, id, comp\rangle$ of collections and functions, where C^o is the collection (not necessarily set) of *objects* in the category and C^a is the collection of arrows. Each arrow a has a unique *domain* and *codomain*, given by $dom(a) : C^a \to C^o$ and $cod(a) : C^a \to C^o$. More specifically, the domain of an arrow is a single object; likewise for the codomain. This conception of (co)domains as single objects can be counter-intuitive, especially in the category of sets: here, the (co)domain of an arrow is a single object that happens to be the set that is the arrow's set-theoretic (co)domain. That is, in the category of sets, these (co)domains are themselves sets, so an arrow/set-theoretic function has a (co)domain two distinct senses. However, in a general setting, the (co)domain of an arrow is simply an object; it is extremely important to remember that the objects of the category need not be collections, and the arrows need not be functions. When the domain of an arrow a is A and the codomain is B, we write this as $a : A \to B$, an unfortunate abuse of notation that reinforces the incorrect notion that arrows are set-theoretic functions. To avoid confusion, we will explicitly note when we are using this notation for arrows or for functions. Note that there many be many arrows $A \to B$. For a category $\mathcal{C}$, this set of arrows is often denoted $\mathcal{C}(A, B)$. Even when the objects themselves are not sets, there can still be multiple arrows between two objects.

The function $id : C^o \to C^a$ gives the *identity* arrow for an object. That is, for $a \in C^o$, $id(a)$ is a map whose domain and codomain are both a. (Many arrows may have domain and codomain a; one must be chosen as the identity, and different choices of identity will yield different categories, at least from an intensional point of view.) Often, $id(a)$ is written as id_a. The function $comp : C^a \times C^a \rightharpoonup C^a$ characterizes function composition: for arrows $f, g \in C^a$, $comp(f, g) = (f \circ g)$, provided that the composition is defined (hence the definition of *comp* as a partial function). We *require* that $comp(f, g)$ be defined whenever $cod(g) = dom(f)$. Note the abuse of the notation $\circ$, which is now used to denote both function and arrow composition. Also note that the composition of two arrows $g : a \to b$ and $f : b \to c$ is an arrow $f \circ g : a \to c$.

Finally, we require two axioms. The first is that identity is a "unit." Let arrow $f : a \to b$. Then it must be the case that $f = id(b) \circ f = f \circ id(a)$. The second is that composition

is associative: Let arrows $h : a \to b$, $g : b \to c$, and $f : c \to d$. We must have that $f \circ (g \circ h) = (f \circ g) \circ h$. Notice that equality of arrows is necessarily defined by the objects within the category. In other words, to determine if two arrows are equal, we must use the definitions given by the objects in the category.

One particularly compact way to represent equations between compositions of arrows is through the use of a commutative diagram. In a diagram, every composition of arrows that begin at the same object and end at the same object are equated. In the following figure, it must be the case that $f \circ g = id_A$. Note, however, that the diagrams *are not* depictions of the category. That is, the category may have arrows f and g such that $f \circ g \neq id_A$. In this case, *these arrows cannot appear in this diagram.* In other words, a diagram is a compact way to express a set of equations, *not* a way of pictorially representing the category. The figure below defines category-theoretic *isomorphism*: when arrows $f : A \to B$ and $g : B \to A$ have the property that $f \circ g = id_A$ and $g \circ f = id_B$, we say that A and B are isomorphic. This definition of isomorphism is more general that our usual set-theoretic notion. In particular, in the category of sets, two sets A and B are isomorphic if they have the same cardinality, because in this case, we know must have at least one one-to-one, onto function from A to B (by the definition of cardinality), and we may find an inverse that is also a one-to-one onto function. Thus, not matter how many times we "cycle" through A and B, we still end up with an identity function.

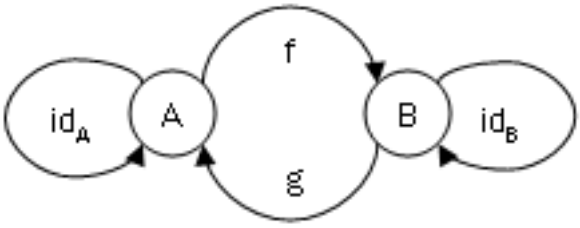

Figure 1: Diagram for Isomorphism

This is all the category theory that is required to understand later category-theoretic concepts. The rest of the paper will build on these definitions to introduce further category-theoretic connections as appropriate. With these preliminaries out of the way, we are now ready to take our first steps toward constructing λ-models.

3 Toward Natural Models

Before we can examine the properties that λ-models should have, we must first define the idea of a *valuation.* Valuations are necessary because not all λ-terms are closed. We desire to assign an interpretation to every term in λ (λ generically refers to either $\lambda\beta$, $\lambda\beta\eta$, or one's favorite lambda calculus), but it is unclear how to interpret variables: after all, they vary.

So, we interpret variables relative to an *environment*; that is, we give them a valuation. Intuitively, a free variable is very similar to a constant symbol. When we construct our interpretation function, we will call upon the environment to give a meaning to a variable. This idea is directly analogous to an environment in first-order logic.

Definition 1. *Let D be a set. Any mapping from the free variables of λ into D is called a valuation, usually denoted by ρ or σ. For $d \in D$ and x a variable, $[d/x]\rho$ denotes the valuation ρ' which is the same as ρ except that $\rho'(x) = d$.*

For example, if $\rho(x) = 3$, then our interpretation of x may involve 3. This notion must necessarily be left imprecise for now, as it is only in connection with applicative structures that valuations become useful and necessary.

3.1 Applicative Structures

Applicative structures are structures that, in a certain sense, embody the notion of *application*. Models of lambda calculus, both typed and untyped, are usually applicative structures. This definition of such a structure is taken from [1], section 10A.

Definition 2. *An applicative structure is a pair $\langle D, \cdot \rangle$, where D is a set, called the domain of the structure, that has at least two members, and $\cdot$ is any mapping from $D \times D$ to D. Intuitively, $\cdot$ "applies" the first argument to the second, yielding a third result element. ($\cdot$ is total in our definition.)*

A good example of an applicative structure is a non-trivial group. Because every element in the group may be "applied" to any other through the group operation, the result is an applicative structure. In fact, this view of an applicative structure as an algebra is both convenient and useful. As it is our intention to build λ-models from applicative structures, and vice-versa, we must study the relationship between arbitrary functions and functions that can be expressed using $\cdot$.

Definition 3. *Let $\langle D, \cdot \rangle$ be an applicative structure and let $n \geq 1$. A function $\theta : D^n \to D$ is representable in D iff D has a member a such that for every $d_1, \ldots, d_n \in D$, $a \cdot d_1 \cdot \ldots \cdot d_n = \theta(d_1, \ldots, d_n)$. Every such a us called a representative of θ. The set of all representable functions from D^n to D is denoted $(D^n \to D)_{rep}$.*

Simply put, a function θ is representable iff there is an element $a \in D$ that a behaves like θ does. (We sometimes say that a and θ *track* each other.) The distinction between representable and non-representable functions is an important one, because in general, there are many more functions $D^n \to D$ than there are functions that can be built using the elements of D and $\cdot$. In keeping with our group theory example, let G be a group; any function $f : G \times G \to G$ that does not obey the group axioms is, by definition, not representable. If it were, we'd no longer have a group!

We may also move from representable functions to functions:

Definition 4. *For every $a \in D$, $Fun(a)$ is the unique one-place function that a represents. That is, for every $d \in D$, $Fun(a)(d) = a \cdot d$. In addition, for $\theta \in (D \to D)_{rep}$, let $Reps(\theta)$ be the set of all of θ's representatives in D.*

If we construct an applicative structure for λ by letting $\cdot$ be the application of λ-terms and have $D = \{$ every λ-term, then one obviously non-representable function is the halting function $halt : D \to D$ given by

$$halt(x) = \begin{cases} \mathbf{0} & \text{if } x \text{ has a normal form} \\ \mathbf{1} & \text{otherwise.} \end{cases}$$

(Here, $\mathbf{0}$ and $\mathbf{1}$ are the Church numerals representing 0 and 1. That is, $\mathbf{0} \equiv \lambda f.\lambda x.x$ and $\mathbf{1} \equiv \lambda f.\lambda x.fx$.) We know that if this function were representable, then there would be some term f such that for every term x, fx reduces to $\mathbf{0}$ if x has a normal form and reduces to $\mathbf{1}$ otherwise. We know that this is impossible by Church's proof.

With this simple definition of an applicative structure we already have properties that will be useful in constructing λ-models. However, we need to examine how other concepts that play key roles in λ-calculus are mirrored in applicative structures. One such concept is *extensionality*.

Definition 5. *Let $a, b \in D$. We say that a is extensionally equivalent to b, denoted $a \sim b$, iff for every $d \in D$, $a \cdot d = b \cdot d$. An applicative structure $\langle D, \cdot \rangle$ is extensional iff for every $a, b \in D$, we have that for every $d \in D$, $a \cdot d = b \cdot d$ implies that $a = b$.*

This is a straightforward definition of extensionality; the general definition of extensionality for functions is usually expressed as, for functions f and g: $\forall x, f(x) = g(x) \iff f = g$. A similar notion is also found in set theory, where the axiom of extensionality states that for sets a and b, $\forall c \in a \wedge c \in b \iff a = b$.

Extensionality in λ-calculus is usually given by η-conversion, which converts between $\lambda x.fx$ and f whenever x is not free in f. At first, this definition is not obviously related to extensionality; however, we can recover a more function-oriented definition of extensionality from this rule by showing first, that if f and g are extensionally equivalent, then η-reduction is valid, and second, that if η-conversion is valid, then we have extensionality. We do this as follows: first, suppose f and g are extensionality equivalent (that is, $fa =_\alpha ga$ for every λ-term a). Let a be a variable x not appearing free in either f or g. Then we have that $fx =_\alpha gx$ and thus that $\lambda x.fx =_\alpha \lambda x.gx$. (The abstraction can be taken on both sides because x is not free.) By η-conversion, we recover $f =_\alpha g$. Secondly, consider a term $(\lambda x.fx)y =_\beta fy$. Here we have that $\lambda x.fx =_\alpha f$, which is the definition of η-conversion. (Note that in this proof we are implicitly using rule ξ.)

We may use extensionality to define equivalence classes:

Definition 6. *Let $a \in D$. Then the extensional equivalence class containing a is the set*

$$\tilde{a} = \{\ b \in D \mid b \sim a\ \}$$

Furthermore, the set of all these classes is denoted by $D/\sim = \{\ \tilde{a} \mid a \in D\ \}$.

The notation $D/\sim$ is highly indicative of the operation of forming a quotient group in group theory. In a sense, $D/\sim$ is D that has been "divided out" by mapping the equivalence classes to single elements, much like the group G/N maps N to 1 by dividing out N.

Applicative structures, which in many ways "feel" like λ-models already, can actually be related to special, stronger algebras known as combinatory algebras.

3.1.1 Combinatory Algebras

Combinatory algebras are essentially applicative structures with a small amount of additional structure. This additional structure is often useful in defining λ-models.

Definition 7. *A (total) combinatory algebra is a pair* $\mathcal{D} = \langle D, \cdot \rangle$*, where* D *is a set with at least two members,* $\cdot : D \times D \to D$*, and* D *has members* k *and* s *such that for all* $a, b, c \in D$*,* $k \cdot a \cdot b = a$ *and* $s \cdot a \cdot b \cdot c = a \cdot c \cdot (b \cdot c)$*. (In a partial combinatory algebra,* $\cdot$ *may be partial.)*

Combinatory algebras are applicative structures that guarantee the existence of elements k and s. This existence guarantee is not an accident: these definitions of k and s are exactly analogous to the combinators **K** and **S**. In fact, that **K** and **S** form a basis for λ is a reflection that combinatory algebras with k and s are *combinatorially complete*.

Definition 8. *An applicative structure* $\langle D, \cdot \rangle$ *is combinatorially complete iff for every sentence* $x_1, \ldots, x_n$*, and every combination* X *of* $x_1, \ldots, x_n$ *only, the formula*

$$\exists u \forall x_1, \ldots, x_n (u x_1 \ldots x_n = X)$$

is true in D*.*

Combinatory completeness intuitively expresses that elements in a sequence $x_1, \ldots, x_n$ may be permuted in any way by choosing an appropriate term u. In λ and combinatory logic, such permutation is usually accomplished by selector and pairing functions like **K**.

It is obvious that every applicative structure that is a combinatory algebra is combinatorially complete. However, we may use the definition of combinatory completeness to define a combinatory algebra without needing to explicitly mention elements k and s.

We are ready to develop our first λ-models. In our development, we will try to use applicative structures rather than combinatory algebras to keep the development as general as possible. However, the use of combinatory algebras is often convenient because it simplifies proofs and the existence of k and s are mirrored in the existence of combinators **K** and **S** in most variants of λ.

4 Our First Models

With these definitions in mind, we may now define properties required of all models of untyped lambda calculus.

Definition 9. *A λ-model or model of untyped lambda calculus is a triple $\mathcal{D} = \langle D, \cdot, [\![\ ]\!] \rangle$, where $\langle D, \cdot \rangle$ is an applicative structure and $[\![\ ]\!]$ is a mapping which assigns, to each lambda term M and each valuation ρ, a member $[\![M]\!]_\rho \in D$ such that*

(a)	$[\![x]\!]_\rho$	$=$	$\rho(x)$	*when x is an atom;*
(b)	$[\![PQ]\!]_\rho$	$=$	$[\![P]\!]_\rho \cdot [\![Q]\!]_\rho$	*for terms P, Q;*
(c)	$[\![\lambda x.P]\!]_\rho \cdot d$	$=$	$[\![P]\!]_{[d/x]\rho}$	*for terms P and every $d \in D$;*
(d)	$[\![M]\!]_\rho$	$=$	$[\![M]\!]_\sigma$	*if $\sigma(x) = \rho(x)$ for every x in the free variables of M;*
(e)	$[\![\lambda x.M]\!]_\rho$	$=$	$[\![\lambda y.[y/x]M]\!]_\rho$	*if y is not free in M;*
(f)	$\forall d \in D, [\![M]\!]_{[d/x]\rho}$	$=$	$[\![N]\!]_{[d/x]\rho}$	*implies $[\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$*

Note that $[\![M]\!]_\rho$ may be written as $[\![M]\!]^{\mathcal{D}}_\rho$, or simply $[\![M]\!]$ when it is independent of ρ.

As discussed in [1], we would intuitively expect each of these conditions to hold in a model of untyped lambda calculus. We must be able to assign an *interpretation* or *meaning* to every lambda term, and a change of valuation essentially creates a new model; hence the inclusion of valuation ρ in the definition.

Clauses (a) and (b) give the meaning of $[\![M]\!]_\rho$ when $M \equiv x$ and $M \equiv PQ$. Clauses (c) through (f) are needed for the case when $M \equiv \lambda x.P$. Clause (d) is a standard property required by first-order model theory and the definition of valuations. Clause (e) corresponds to the axiom scheme α, and is needed so that terms congruent by α-conversion have the same interpretation. Clause (f) similarly is required for the rule ξ, which says that $X =_\beta Y$ implies $\lambda x.X =_\beta \lambda x.Y$, which holds because any contraction or change of bound variables made in X can also be made in $\lambda x.X$. Finally, clause (c) expressed the intuition behind lambda abstraction: it says that $[\![\lambda x.P]\!]_\rho$ acts like a function whose value is calculated by interpreting x as s.

One consequence of these properties is called *weak extensionality*. This property is that

$$[\![\lambda.M]\!]_\rho \sim [\![\lambda x.N]\!]_\rho \text{ implies } [\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$$

Intuitively, this states that terms in the same equivalence class have the same interpretation. Probably the most important consequence of these properties (a)-(f) is that every λ-model satisfies all provable equations of $\lambda\beta$. Here, satisfaction is defined as usual:

$$\models_{\mathcal{D},\rho} M = N \text{ iff } [\![M]\!]_\rho = [\![N]\!]_\rho, \text{ and}$$

$$\models_{\mathcal{D}} M = N \text{ iff } \models_{\mathcal{D},\rho} M = N \text{ for every } \rho,$$

Of course, our λ-model wouldn't be much of a model if it didn't satisfy all the provable equations. It is an interesting question to ask if, in a λ-model, there are true equations that are not provable.

For instance, an *unsolvable* term M is a term without a normal form such that there are no terms N_n for which

$$MN_1 \ldots N_n = I$$

All of these terms may be equated by an axiom schema and the resulting theory will still be consistent and non-trivial. Axioms (a)-(f) do not dictate whether or not all unsolvable terms have the same interpretation. We may take this as an indication of the richness of the model theory for λ.

Another useful result of these axioms is that if $\langle D, \cdot, [\![\ ]\!] \rangle$ is a λ-model, then $\langle D, \cdot \rangle$ is a combinatory algebra. In a certain sense, this is to be expected, because applicative structures that contain k and s are combinatory algebras, and we may construct k and s using the axioms of λ.

We may also define models of lambda calculus with η reduction: a model of $\lambda\beta\eta$ is a λ-model that satisfies the equation $\lambda x.Mx = M$ for every M with x not free. From this, we may obtain that a λ-model us extensional iff it is a model of $\lambda\beta\eta$.

Now that we know what properties our models must have, we may start to construct our first models. Our first models are called term models, because they are constructed out of the syntax of λ-terms.

4.1 Term Models

One easy to construct, but somewhat unsatisfying class of models are called *term models* of $\lambda\beta$ or $\lambda\beta\eta$. Let Γ be $\lambda\beta$ or $\lambda\beta\eta$. For each lambda term M, define

$$[M] = \{\ N \mid \Gamma \vdash M = N\ \}$$

The *term model* of Γ is $\langle D, \cdot, [\![\ ]\!] \rangle$, where $D = \{\ [M] \mid M \text{ is a term}\ \}$, $[M] \cdot [N] = [MN]$, and $[\![M]\!]_\rho = [\ [N_1/x_1, \ldots, N_n/x_N]M\ ]$, where $\{x_1, \ldots, x_n\}$ are the free variables of M and $[N_1/x_1, \ldots, N_n/x_N]$ is simultaneous substitution.

In this model, a term is interpreted as the equivalence class of terms that are provably equal to it. Thus, it is obvious that this model satisfies every provable equation. In fact, it is too obvious: we gain very little insight from such a model. The situation is not unlike that of term models for FOL that are used in model existence proofs – term models demonstrate that models exist, but give us no other insight. We can try, with a debatable amount of success, to remove the heavy reliance on syntax.

4.1.1 Syntax-free Models

We can remove the reliance of term models on syntax by focusing on extensional equivalence classes instead of provability.

Definition 10. *A syntax-free λ-model is a triple $\langle D, \cdot, \Lambda \rangle$, where $\langle D, \cdot \rangle$ is an applicative structure, Λ is a map from D to D, and the following four properties hold:*

1. *$\langle D, \cdot \rangle$ is combinatorially complete,*
2. *for every $a \in D$, $\Lambda(a) \sim a$,*

3. *for every* $a, b \in D$, $a \sim b$ *implies* $\Lambda(a) = \Lambda(b)$,

4. *there exists an* $e \in D$ *such that for every* $a \in D$, $e \cdot a = \Lambda(a)$.

Rather than letting a term be interpreted as an equivalence class of the terms provably equal to it, here we use Λ to select a *representative* for each extensional equivalence class and interpret a term as its representative. As before, it is obvious that provably equal terms, being in the same extensional equivalence class, will have the same interpretation. The "advantage" of this definition is that it only deals with extensionality and does make use of syntactic notions of provability; hence the "syntax-free" moniker. However, extensionality and provable equality are intimately related, and there is an interesting connection between syntax-free and regular λ-models: $\langle D, \cdot, \Lambda \rangle$ is a syntax-free λ-model iff $\langle D, \cdot, [\![\ ]\!] \rangle$ is a λ-model with $[\![\ ]\!]$ is defined as

$$\begin{array}{lcll} [\![x]\!]_\rho & = & \rho(x) & \\ [\![PQ]\!]_\rho & = & [\![P]\!]_\rho \cdot [\![Q]\!]_\rho & \\ [\![\lambda x.P]\!]_\rho & = & \Lambda(a) & \text{where for every } d \in D, a \cdot d = [\![P]\!]_{[d/x]\rho} \end{array}$$

(This is just a formalization of what we would expect.) Conversely, we may define Λ in terms of $[\![\ ]\!]$ by

$$\Lambda(x) = [\![\lambda x.ux]\!]_{[a/u]\sigma} \text{ for any } \sigma$$

(This just formalizes that any representative of the equivalence class may be chosen.)

Technically speaking, the constructions of $[\![\ ]\!]$ and Λ are mutual inverses, in the sense that if we let $\langle D, \cdot, [\![\ ]\!]' \rangle$ be a λ-model, and we define Λ from $[\![\ ]\!]'$, and then define $[\![\ ]\!]$ from Λ, we will find that $[\![\ ]\!] = [\![\ ]\!]'$. Conversely, if we start with a syntax free model $\langle D, \cdot, \Lambda' \rangle$ and define first $[\![\ ]\!]$ and then Λ, we get $\Lambda = \Lambda'$.

Syntax free models are a bit more useful than terms models because they shed some light on a useful construction called a retraction.

4.2 Retractions

The concept of a *retraction* may be the single most important concept in the denotational semantics of the untyped lambda calculus. It is retraction that allows us to find a domain D such that $D \cong D \to D$. A retraction is a set that may be embedded into another in a special way.

Definition 11. *Let* C *and* D *be sets, and let* I_C *denote the identity function on* C. *Let the functions* $\phi : C \to D$ *and* $\psi : D \to C$ *satisfy* $\psi \circ \phi = I_C$. *Then* ψ *is called a left inverse of* ϕ *and* C *is called a retract of* D *by* ϕ *and* ψ. *The pair* $\langle \phi, \psi \rangle$ *is called a retraction.*

The name "retraction" may be justified by observing that D may be retracted into C via ψ in such a way that C's image under ϕ is preserved. Basic properties of ϕ and ψ include that ψ is onto and that ϕ is one-to-one. We say that ϕ is an embedding of C onto a subset of D, and that C is a retract of D.

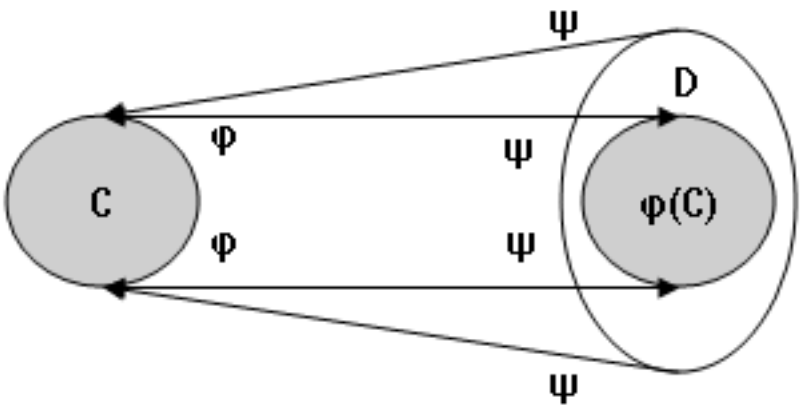

Figure 2: Retraction

4.2.1 Syntax-free models as Retractions

We can obtain a very algebraic feeling syntax-free λ-model as follows. Let $\langle D, \cdot, \Lambda \rangle$ be a syntax free λ-model. Recall that $(D \to D)_{rep}$ is the set of all representable one-place functions, $Reps(\theta)$ is the set of all representatives of a function $\theta \in (D \to D)_{rep}$, and $Fun(a)$ is the one-place function represented by $a \in D$. Note that $Reps(\theta)$ may have many members, but Λ gives us a way of singling one out as the representative of the equivalence class. We call this $Rep(\theta) = \Lambda(a)$, for any $a \in Reps(\theta)$. Thus, we have that $Fun(Rep(\theta)) = \theta$. Therefore Fun is a left inverse of Rep, and so Rep is a one-to-one embedding of $(D \to D)_{rep}$ into D, and so $(D \to D)_{rep}$ is a retract of D. This is best illustrated in the figure. (Note, however, that unlike in the previous diagram, here it is the right side that is the retract.)

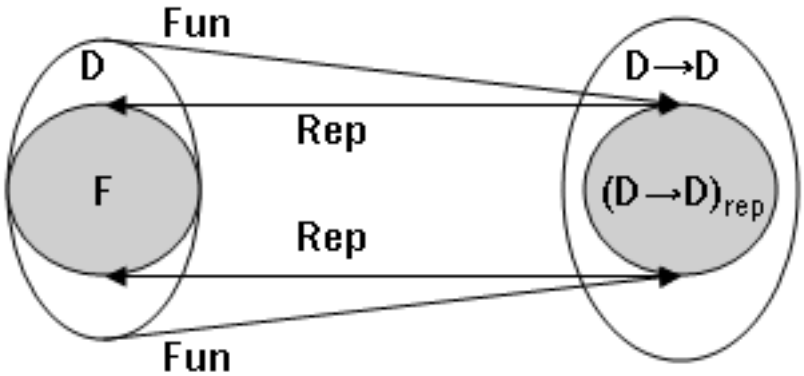

Figure 3: Syntax-free retractions.

We may also reverse this discussion to define application in terms of representability. Let D be a set, and C any set of one-place function from D to D. Let $Rep : C \to D$ and

$Fun : D \to C$ be any pair of functions that form a retraction (specifically, $Fun \circ Rep = I_C$). Then we can define application as follows: for every $a, b \in D$, let $a \cdot b = (Fun(a))(b)$. This definition of $\cdot$ has the property that C becomes exactly the set of all functions representable in $\langle D, \cdot \rangle$. That is, it becomes the gray circle on the right side of the diagram. If we take $\Lambda = Rep \circ Fun$, then we have a λ-model. In other words, given any term in D, to interpret it we "project" it into the set of representable functions, and then "retract" it back by finding its representative.

Admittedly, using Λ, *Rep*, and Fun in this way resembles a dog chasing its tail. However, this game of circularity yields an extremely important result: any retraction where D has at least two members, $\cdot$ is defined as above (and hence $\langle D, \cdot \rangle$ is combinatorially complete), and $Rep \circ Fun \in C$ gives rise to a λ-model. (Bear in mind that D may have far few elements than we would expect, and this it becomes a λ-model through combinatory completeness, rather than by having a domain that "feels right.")

Retractions are only the first concepts needed on the road to natural λ-models.

5 CPOs and Scott's Topology

Complete partial orders and Scott's topology (named after Dana Scott, who essentially invented the concept in the 1970's) the are key to understanding realistic models of the untyped lambda calculus. That is, we may use the concepts to construct, from the ground up, more semantic feeling λ-models. At first, these definitions may seem like they have little to do with lambda calculus; indeed, many of these definitions have a strong relationship to topology. (For instance, consider topological spaces $A \subseteq X$ and let $r : X \to A$ be a continuous map whose restriction to A is the identity map on A. In this case, A is a retract of X.) These definitions are taken from [1]; other, slightly different definitions are available in related texts. Interestingly, many of the concepts introduced in this section are more intuitive in a typed setting. See, for instance, [4]. The foundational concept is that of a partially ordered set:

Definition 12. *A partially ordered set, or poset, is an ordered pair $\langle D, \sqsubseteq \rangle$ where D is a set and $\sqsubseteq$ is a transitive, anti-symmetric and reflexive binary relation on D. If D has a least element, it is called bottom, and is denoted by $\bot$. When $a \sqsubseteq b$, we say that a is less than or equal to b, or that b is greater than or equal to a.*

Recall that a transitive relation $\sqsubseteq$ has the property that $\forall a, b, c, a \sqsubseteq b \longrightarrow b \sqsubset c$. Reflexivity implies that $\forall a, a \sqsubseteq a$. Finally, antisymmetry implies that $\forall a, b, a \sqsubseteq b \longrightarrow \neg(b \sqsubseteq a)$.

Partially ordered sets capture the notion that some elements are greater than others; however, any given element is not necessarily comparable to any other. As such, a poset is a much like a set of ascending chains. When the poset has a least element, then all the chains must be connected to $\bot$, and so the poset resembles a tree. Posets are most often

used to express that some functions are "more defined" than others; this use will come in handy later. We now proceed to examine posets themselves.

Definition 13. *Let $\langle D, \sqsubseteq \rangle$ be a poset. An upper bound of a subset $X \subseteq D$ is any $b \in D$ such that $a \sqsubseteq b$ for any $a \in X$. The least upper bound, or lub, of X is denoted $\sqcup X$ and is $\sqsubseteq$ any upper bound of X. For arbitrary X, $\sqcup X$ may not necessarily exist or be in X, but if it does exist, then it is unique.*

The uniqueness easily follows from reflexivity, transitivity, and antisymmetry. If $\sqsubseteq$ is used to order functions by degree of definedness, the lub of a directed set is often used to denote a "completely defined" function.

Definition 14. *Let $\langle D, \sqsubseteq \rangle$ be a poset. A subset $X \subseteq D$ is directed if X is non-empty and*

$$\forall a, b \in X, \exists c \in X \text{ s.t. } a \sqsubseteq c \text{ and } b \sqsubseteq c$$

A good intuition that captures the essence of directedness, and also demonstrates why the word "directed" was chosen, is this: imagine the set X laid out as a series of pegs on a pegboard, where pegs that are related by $\sqsubseteq$ are connected by strings, with the greater element to the right of the smaller element. Pick two pegs a and b. There must be a c to the right of them both that you can get to by following strings from a and b. Then, pick another element d. There must be a peg to the right of both c and d that you can get to by following strings connected to c and d. Then, pick another element d, and so on. Through this process we are moving steadily to the right; hence, a directed set X somehow directs you to greater and greater elements. If this traveling stops, then $\sqcup X$ exists.

Definition 15. *A complete partial order, or cpo, is a partially ordered set $\langle D, \sqsubseteq \rangle$ such that D has a least member $\bot$ and every directed $X \subseteq D$ has a least upper bound $\sqcup X$. Usually we will speak of the cpo D rather than the cpo $\langle D, \sqsubseteq \rangle$ because the $\sqsubseteq$ in question is obvious from context. Note that a set D may have many such $\sqsubseteq$s.*

In a certain sense, these partial orders are considered complete because there are no "missing" limits (either $\bot$ or any lub of a directed set.) The existence of these limits has certain nice properties that we will use when we construct our models. (In fact, every cpo that has a continuous function space that can be embedded in it via a retraction is a λ-model in which exactly the representable functions are continuous, and vice versa.)

One very common way to construct cpos from sets is via *lifting*:

Definition 16. *Let D be a set. Then the cpo $\langle D_\bot, \sqsubseteq \rangle$, called "D lifted," is constructed by taking $D_\bot = D \cup \{\bot\}$, where $\bot$ is some distinguished element such that $\bot$ is less than every element of D. The elements of $D \subset D_\bot$ that are not $\bot$ are not related by $\sqsubseteq$ at all. A good intuition may be found in the example in following picture:*

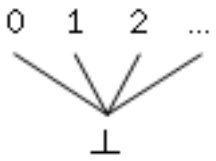

Figure 4: $\mathbb{N}_\perp$

The reason that lifting is useful is because it is commonly used to transform partial functions in to total ones. For instance, let $f : \mathbb{N} \rightharpoonup \mathbb{N}$. Then, for any $x \in \mathbb{N}$ that is not in the domain of f, define $f'(x) = \perp$ and let f' agree with f on all other values. Then $f' : \mathbb{N} \to \mathbb{N}_\perp$ is a total function, and we have that $f'(x) \sqsubseteq f'(y)$ whenever $x \sqsubseteq y$. In other words, we use $\perp$ and $\sqsubseteq$ to express that "undefinedness" is less than "definedness." We will formalize this intuition later; for now, we have stumbled upon an important property:

Definition 17. *Let $\langle D, \sqsubseteq \rangle$ and $\langle D', \sqsubseteq' \rangle$ be cpos and let $\phi : D \to D'$. We say ϕ is monotonic iff for all $a, b \in D$, $a \sqsubseteq b$ implies $\phi(a) \sqsubseteq' \phi(b)$. We say ϕ is continuous iff for all directed $X \subseteq D$, $\phi(\sqcup X) = \sqcup(\phi(X))$.*

The intuition behind monotonicity is the same as in other fields of mathematics: if we arrange the preimage and imagine of a monotonic function in ascending chains, and connect the elements related by the map, then none of the connections will cross each other.

One trivial but important property of these definitions is that every continuous function is monotonic. In many respects, there is no simple intuition that captures why this is true, but we can see that if a function ϕ maps a lub to a non-lub, then the ordering on D' has been violated by ϕ, either by breaking monotonicity or by having extra elements below where the lub "should be." Note that not every monotonic function is continuous. We say that a function that has $f(\perp) = \perp$ is *strict*. (The class of strict functional programming languages are called strict because their functions behave in this manner. In addition, a kind of $\perp$ has made it into programming languages via the type *unit*.)

It is possible to characterize the Scott topology in purely topological terms, for instance, defining open sets. However, that approach requires an understanding of topology.

An extremely interesting and important result from denotational semantics is that continuous functions on cpos are computable in a recursion-theoretic sense. A related fact is that the functions always have fixed points. We will not diverge into this intriguing subject here; instead, we will examine the connection of cpos to category theory.

5.1 Connections to Category Theory

There are many natural connections between cpos and category theory. For instance, the class of cpos forms a category, called *Cpo*, where the objects are cpos and the arrows are continuous functions between cpos (objects). (In some texts, for instance [4], our cpos are

considered *pointed*, because they contain $\perp$. In this case, the pointed cpos form a subcategory of *Cpo*.).

In addition, a single partial order $\langle D, \sqsubseteq \rangle$ is a category: the objects are the elements of D and two objects a, b have an arrow from $a \to b$ iff $a \sqsubseteq b$. Note that here there is necessarily at most one arrow between any two objects. Also note that the identity function exists because $a \sqsubseteq a$ for every $a \in D$, and that *comp* exists because a partial order is transitive. A partial order is a good example of where set-theoretic intuition can fail us, because the arrows in a partial order category aren't functions, and so it seems like the category is nothing more than a directed, and possibly disconnected, graph.

Another particularly illustrating connection is that both lifting and product construction are *functors*.

Definition 18. *Let C and D be categories. A functor from C to D is a pair of maps $\langle F^o, F^a \rangle$ with the following properties:*

- $F^o : C^o \to D^o$, *that is, F^o maps objects to objects*
- $F^a : C^a \to D^a$, *that is, F^a maps arrows to arrows*
- *whenever $f : a \to b$, we have $F^a(f) : F^o(a) \to F^o(b)$*
- $F^a(id_C(a)) = id_D(F^o(a))$ *for every $a \in C^o$, that is, F^a maps identity to identity*
- $F^a(f \circ g) = F^a(f) \circ F^a(g)$ *for every $f, g \in C^a$.*

We usually omit the superscripts from functors when no confusion is possible. Intuitively, functors are functions from categories to categories that preserve structure. In a sense, functors are the categorical equivalents of homomorphisms.

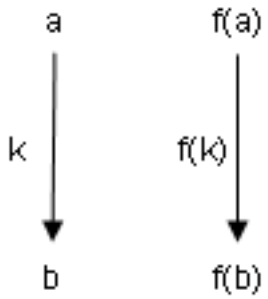

Figure 5: A Functor f.

Lifting is a functor from the category of non pointed (without $\perp$) cpos to *Cpo*; the object map takes a non pointed cpo D to the cpo $D_\perp \in Cpo$, and the arrow map takes a continuous function $f : A \to B$ to the strict continuous function $f_\perp : A_\perp \to B_\perp$. The product functor

is a functor from $Cpo \times Cpo$ to Cpo. The object map takes a pair of cpos $\langle C, D\rangle$ and maps them to the product CPO $C \times D$, whilst the arrow map takes a pair $\langle f, g\rangle$ of continuous functions with $f : C \to C'$ and $g : D \to D'$ and maps them to the continuous function $f \times g : (C \times D) \to (C' \times D')$ such that $(f \times g)(\langle x, y\rangle) = \langle fx, gy\rangle$.

The formation of a continuous function space is also a functor, although its construction is slightly more involved than the previous examples – it involves working with a construction called an *opposite category*, and it will not be given here.

5.2 Cartesian Closed Categories

A Cartesian closed category (ccc) is a special kind of category that is a suitable structure for capturing pairing and function application; in fact, much of the motivation for using category theory is to have a common framework for working with constructions that are useful in constructing λ-models. One consequence of this natural connection between λ-models and cccs is that λ-models are quite often cccs, and vice-versa.

To give a definition of a ccc, we must first define how products and functions are represented inside of categories.

Definition 19. *Let a, b, and c be objects of a category. We say that $[a \times b]$ is the product of a and b if there are arrows $p_1 : [a \times b] \to a$ and $p_2 : [a \times b] \to b$ such that for every arrow f, g as indicated below, there is a unique h such that the following diagram commutes.*

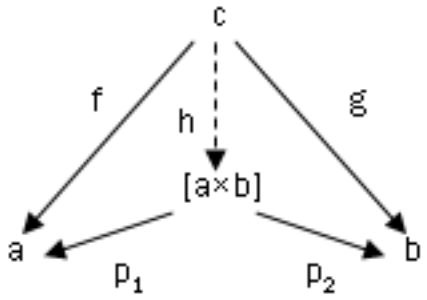

Figure 6: Product formation inside a category.

In this figure, we have adopted the common convention that existentially qualified arrows are denoted with dashed lines. Intuitively, p_1 and p_2 are the projection functions for the pairing constructor h. (Note that square brackets are now used in a categorical settings for pairing and functions, and also for function space construction. The two uses are not related. The square brackets are often used to help remind the reader that the $\times$ is inside the category, and does not denote cartesian product; when there is no danger of confusion, the square brackets are often dropped.)

The second definition we need is that of a *function object* (also called an *exponential object*). This definition makes use of our previous definition of $\times$.

Definition 20. *Let a, b, and c be objects in a category. Then $[a \to b]$ is a function object of a and b when there is an arrow apply : $[a \to b] \times a \to b$ such that for every $f : c \times a \to b$ as indicated below, there is a unique h such that the following diagram commutes.*

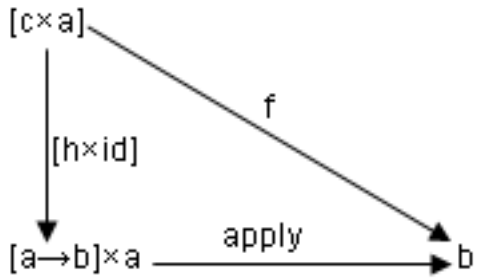

Figure 7: Function formation inside a category.

Intuitively, h is the curried form of f. That is, by using h on c, we obtain a function $[a \to b]$, that, when applied to a via *apply*, yields b. f itself operates on $c \times a$ to yield b. It is perhaps not obvious, but it can be verified that all the function objects of a given pair of objects are isomorphic.

With these definitions in hand, we may proceed to the definition of ccc.

Definition 21. *A category $\mathcal{C}$ is cartesian closed when it has a specified object unit and specified binary maps $\times$ and $\to$ on objects such that for all objects a, b, and c,*

$$\mathcal{C}(a, unit) \text{ has exactly one arrow,}$$

$$\mathcal{C}(a, b) \times \mathcal{C}(a, c) \cong \mathcal{C}(a, b \times c) \text{ and}$$

$$\mathcal{C}(a \times b, c) \cong \mathcal{C}(a, b \to c)$$

(Technically, there are a few conditions on the isomorphism above, but they are "natural" and as such, we will ignore them in the following discussion.)

Simply put, a ccc is a category with a special object *unit* that "contains" exactly one element. However, because we cannot treat the elements in the category as collections, we must state this condition by saying that $\mathcal{C}(a, unit)$ has exactly one arrow, which is an equivalent statement in most non-pathological categories. The intuition here is that if there is only map from b to a, then a must only have one element – this is particularly easy to see in the category of sets, where for every object (set), there can be only one arrow into *unit*; namely, the constant function whose range is *unit*.

In a similar way, the condition $\mathcal{C}(a, b) \times \mathcal{C}(a, c) \cong \mathcal{C}(a, b \times c)$ uses this method of "counting" to express that $b \times c$ is the collection of pairs of elements drawn from b and c. Intuitively, we "fix" an a and then we can pull the cartesian product into the category itself. The final axiom, that $\mathcal{C}(a \times b, c) \cong \mathcal{C}(a, b \to c)$, states that $b \to c$ is the collection of all arrows from b to c. This is best illustrated by the example of $a = unit$, that $\mathcal{C}(unit \times b, c) \cong \mathcal{C}(unit, b \to c)$.

Since *unit* is a one "element" collection, we have that $b \cong unit \times b$. So, $\mathcal{C}(unit \times b, c) \cong \mathcal{C}(b, c)$. From here we have that $\mathcal{C}(unit, b \to c) \cong \mathcal{C}(b, c)$. In other words, we are saying that $b \to c$ captures all the arrows from b to c inside the category, in much the same way that $\times$ captures cartesian product inside the category.

Many of the structures we have defined earlier have direct analogs in cccs. For instance, we may construct a term ccc in much the same way as we construct a term model. In addition, the category $Cpo_\perp$ of (pointed) cpos and strict continuous functions is cartesian closed.

There are two extremely important connections with category theory:

1. Given a ccc, we can recover an applicative structure, and

2. Given an applicative structure, we can construct a ccc.

It is this equivalence that allows us to move freely from category theoretic investigations into a more classical setting, and vice versa, allowing the full power of both approaches to be harnessed. In fact, a large number of "categorical semantics" of λ have been developed [5]. (Of course, there are a few conditions on the above connections. Not just any applicative structure or ccc will do.)

With the connections to category firmly established, we may now investigate the first natural λ-model, D_∞, itself.

6 D_∞

6.1 Preliminaries

Dana Scott's model D_∞ was the first λ-model to be constructed in a way that captures our intuition of λ-terms as functions on themselves. Because the set-theoretic construction of a function space $D \to D$ from a domain D results in a function space with strictly greater cardinality than D, more sophisticated mathematical techniques were required to both determine what the domain of the model was and to characterize an adequate subset of the function space construction that would not create cardinality concerns.

Definition 22. *Let $\langle D, \sqsubseteq \rangle$ and $\langle D', \sqsubseteq' \rangle$ be cpos. The function space $[D \to D']$ is defined to be the set of all continuous functions from D to D'. Furthermore, let $\phi, \psi \in [D \to D']$. We define a partial order $\leq$ based on $\sqsubseteq'$ such that $\phi \sqsubseteq \psi$ iff for every $d \in D$, $\phi(d) \sqsubseteq' \psi(d)$. Note that $\langle [D \to D'], \leq \rangle$ is a cpo, with a least element $\perp$ with the property that for every $d \in D$, $\perp(d) = \perp'$.*

Intuitively, we can think that $a \sqsubseteq b$ means that a is less well defined than b, or that a gives "less information" than b. In the familiar terms of $\mathbb{N}_\perp$, this just means that numbers are more well defined than $\perp$, which can be thought of as non-termination. In keeping with

this way of thinking, we say that $\phi \leq \psi$ whenever for each input value, ϕ yields a less informative answer than ψ.

It is often useful to know that when $D = D'$, the identity function I_D is in the function space $[D \to D]$.

Of paramount importance to the construction of D_∞ is that compositions of continuous functions are continuous. That is, let D, D', D'' be cpos. Then for every $\phi \in [D \to D']$ and $\psi \in [D' \to D'']$, we have that $\psi \circ \phi \in [D \to D'']$. That composition behaves this way provides a sense of closure that will be useful in reasoning about function spaces in general.

Definition 23. *Let D and D' be cpos and let $\phi \in [D \to D']$ and $\psi \in [D' \to D]$ satisfy $\phi \circ \psi = I_{D'}$ and $\psi \circ \phi = I_D$. In this case, we say that D is isomorphic to D', written $D \cong D'$.*

Definition 24. *Let D and D' be cpos. A projection from D' to D is a pair of functions $\phi \in [D \to D']$ and $\psi \in [D' \to D]$ such that $\psi \circ \phi = I_D$ and $\phi \circ \psi \leq I_D$. When such ϕ, ψ exist, we say that D' is projected onto D by $\langle \phi, \psi \rangle$.*

Intuitively, every projection is a retraction, but with the added property that ϕ, ψ are continuous and that $\phi \circ \psi \leq I_D$. Therefore, ϕ, ψ make D isomorphic to a subset $\phi(D)$ of D'. In addition, the bottom members of D and D' correspond: $\phi(\perp) = \perp'$ and $\psi(\perp') = \perp$.

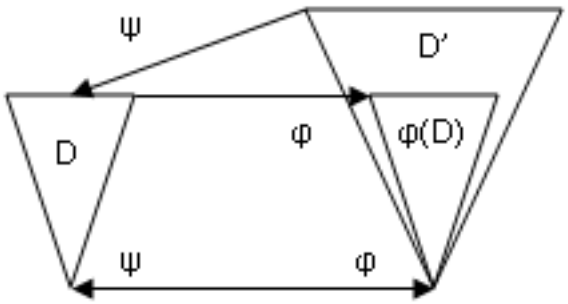

Figure 8: Projection.

6.2 The Intuition

D_∞ is the limit of a sequence $D_0, D_1, \ldots$ of cpos, each of which is the function space of the one before it.

Definition 25. *Let $D_0 = \mathbb{N}_\perp$, and let $D_{n+1} = [D_n \to D_n]$. We will generically refer to $\sqsubseteq_{D_n}$ as $\sqsubseteq$ and refer to the least member of D_n as $\perp_n$.*

We cannot construct a λ-model $\langle D_\infty, \cdot, [\![\ ]\!] \rangle$ in set theory with $\cdot$ as function application and D_∞ a set of functions, because in set theory, a function cannot be applied to itself. (There are λ-models constructed in non-well-founded set theory, but we ignore them here.)

Dana Scott avoided this difficulty by using an intuitively simple device. That is, the members of D_∞ are not functions, but are infinite sequences of functions:

$$\phi = \langle \phi_0, \phi_1, \ldots \rangle$$

where $\phi_n \in D_n$. Application can then be defined as

$$\phi \cdot \psi = \langle \phi_1(\psi_0), \phi_2(\psi_1), \ldots \rangle$$

And self-application becomes possible. As we increase n, we would like that our construction of D_n gives us that $D \cong [D \to D]$. Therefore, we must now formalize the notion of a limit to find D_∞.

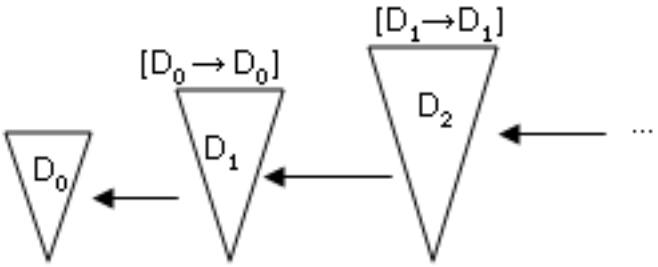

Figure 9: To D_∞.

6.3 The Construction

To define the proper limit we must figure out how to embed each D into the next "larger" D. To do so we must first define a bit of notation.

Definition 26. *We wish to describe functions using the convenient lambda notation, but without the possibility of confusion with λ-terms. So, let Λ be a "meta" lambda. That is, $\lambda x.x$ is a syntactic term, whereas $\Lambda x.x$ is the identity function, whose domain we have not yet defined.*

This notation is simply a convenient device that allows us to quickly and intuitively express functions. The first functions we need to define D_∞ are the map ϕ_0 and the reverse map ψ_0, which embed D_0 into D_1 and project D_1 onto D_0. (The symbol Λ was also used to determine a representative for an equivalence class. It is important not to confuse these two uses.)

Definition 27. *The initial maps ϕ_0 and ψ_0 are defined as follows:*

$$\phi_0(d) = \Lambda a \in D_0.d \text{ for every } d \in D_0$$

$$\psi_0(g) = g(\bot_0) \text{ for every } g \in D_1$$

For every $d \in D_0$, $\phi_0(d)$ is simply the constant function with value d; note that this function is continuous, and so $\phi_0(d) \in D_1 = [D_0 \to D_0]$. Note also that ψ_0 is also continuous and hence also in $[D_0 \to D_0]$. Mapping a constant to the corresponding constant function seems like an intuitive way to embed a set of elements into its function space; likewise, the reverse map is a convenient way to "collapse" a function space by mapping each function to its value at a particular point. In a certain sense, as we move up the D_n, we are "nesting" λ-terms an additional time at every move upward (n to $n+1$), and are peeling away the nesting at every move downward ($n+1$ to n).

It is relatively straightforward to show that $\langle \phi_0, \psi_0 \rangle$ is a projection from D_1 to D_0. We have embedded D_0 into D_1; hence we have a projection from D_1 to D_0.

We can use this projection to "induce" a projection $\langle \phi_n, \psi_n \rangle$ from D_{n+1} to D_n in a natural way. That is, we will have $D_n \leftarrow [D_n \to D_n]$, where $\leftarrow$ denotes projection. In the limit, projection will become isomorphism, and we will have that $D_\infty \cong [D_\infty \to D_\infty]$, which is exactly the property we need.

Definition 28. *Let $n \geq 1$. For every $f \in D_n$ and $g \in D_{n+1}$, we define*

$$\phi_n(f) = \phi_{n-1} \circ f \circ \psi_{n-1}$$

$$\psi_n(g) = \psi_{n-1} \circ g \circ \phi_{n-1}$$

The pair $\langle \phi_n, \psi_n \rangle$ is a projection from D_{n+1} to D_n. The proof is rather involved; we omit it here.

Intuitively, we are trying to figure out how to treat f, which "lives" in D_n, as something "one level higher" so that it may operate on members of D_n. To do this, we take an argument to f in D_n, drop it down into D_{n-1}, apply f, and the bump up the result to D_n. This process is a function from $D_n \to D_n$, and hence is f as it appears in D_{n+1}.

To get a feel for what is going on, first notice that the "cardinality" of each of ϕ_n, ψ_n, f, g is correct. That is, we have $f \in D_n : D_{n-1} \to D_{n-1}$, $g \in D_{n+1} : D_n \to D_n$, $\phi_n : D_n \to D_{n+1}$, and $\psi_n : D_{n+1} \to D_n$. So, we can check to make sure that our compositions are correct: ϕ_n should return something in D_{n+1}, and suppose we supply some $x \in D_n$ to ϕ_n; then the compositions collapse as follows:

1. We apply ψ_{n-1} to x to get something in D_{n-1}.
2. Then, we apply f to the result in D_{n-1} to get an element in D_{n-1}.
3. From here, we apply ϕ_{n-1} to finally get something in D_n. The whole process is then a function from D_n to D_n, and hence the whole process is an element in D_{n+1}. The whole process is how f is embedded into D_{n+1}.

It is certainly possible to formalize the above discussion, but it is more important to understand that ϕ_n is mapping an element f in D_n into the function space D_{n+1} in a very special way. We cannot use f on elements of D_n directly, because f is a function from $D_{n-1} \to D_{n-1}$.

So, we first use ψ_{n-1} to "drop down" arguments to f into D_{n-1}. In this way, elements in D_n are now considered in D_{n-1} and so f may apply to them. From there, we then apply ϕ_{n-1} to "bump up" the result of f back into D_n. The reverse map ψ_n is defined similarly.

This definition of how to move functions from one D_n to another D_{n+1} preserves application in an important sense. Let $a \in D_{n+1}$ and $b \in D_n$. Then

$$\begin{array}{ll} \psi_n(a)(\psi_{n-1}(b)) \sqsubseteq \psi_{n-1}(a(b)) & \text{if } n \geq 1 \\ \phi_n(a(b)) = \phi_{n+1}(a)(\phi_n(b)) & \text{if } n \geq 0 \end{array}$$

Intuitively, this means that if we have one element applied to another, if we "bump up" or "drop down," then the result "does not produce more than we started with" (if we go down) and "acts the same" (if we go up). The intuition is that moving through the D_n does not affect how functions "work."

By using induction beginning with D_0, it is possible to show that certain functions exist in particular D_n. For instance, let $n \geq 2$. Then D_n contains the analog of **K**: $\Lambda a \in D_{n-1}.\Lambda b \in D_{n-2}.\psi_{n-2}(a)$. When $n \geq 3$, we can show that an analog of **S** exists. Having these analogs of combinators will be helpful in establishing that D_∞ is, in fact, a λ-model.

One easy extension is that we may define $\phi_{m,n}$ as the "up or down" function from D_m to D_n in a straightforward way. (That is, we move from D_m to D_n, going either up or down, as required.) This is mostly a convenience.

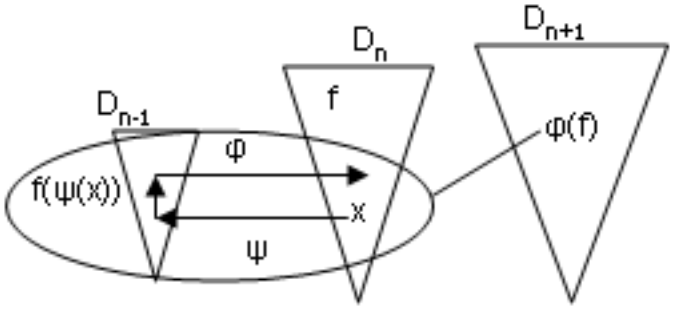

Figure 10: Projection.

6.4 D_∞

We are now in a position to define D_∞. That is, D_∞ is the set of all infinite sequences

$$d = \langle d_0, d_1, \ldots \rangle$$

such that $d_n \in D_n$ and $\psi_n(d_{n+1}) = d_n$ for every $n \geq 0$. Note that here we are saying that each d_n is the result of a projection from D_{n+1}, rather than saying that each d_{n+1} is the result of an embedding from D_n. That is, we are using ψ rather than ϕ. If instead we tried to use ϕ, we would be limited having only elements in D_∞ that could be built up using D_0

rather than having all elements that can be projected down to D_0; these are two different sets of element. With this in mind, let $d, d' \in D_\infty$ and define

$$d \sqsubseteq d' \text{ iff for every } n \geq 0, d_n \sqsubseteq d'_n$$

Intuitively, this means that D_∞ is the collection of sequences where each member of the sequence is the projection of the function after it. $\sqsubseteq$ is intuitively capturing that one element is less than another when this is true piecewise. From this construction we can begin to examine the structure of D_∞. One particularly important property is that D_∞ is a cpo with least member $\bot\langle\bot_0, \bot_1, \ldots\rangle$. Another definition comes in handy here:

Definition 29. *Let $X \subseteq D_\infty$. Define $X_n = \{\ a_n : a \in X\ \}$, where a_n is the n-th member of the sequence a.*

We have that when $X \subseteq D_\infty$ is directed, $\sqcup X = \langle \sqcup X_0, \sqcup X_1, \ldots\rangle$.

Finally, for every n, we can embed D_n into D_∞, and project D_∞ onto D_n, as follows. We can define the mappings $\phi_{\infty,n} : D_\infty \to D_n$ and $\phi_{n,\infty} : D_n \to D_\infty$ as, for every $d \in D_\infty$ and every $a \in D_n$,

$$\phi_{\infty,n}(d) = d_n, \text{ and}$$

$$\phi_{n,\infty}(a) = \langle \phi_{n,0}(a), \phi_{n,1}(a), \ldots\rangle$$

(Notice that when $a \in D_n$, we have that $\phi_{n,n}(a) = a$.) We have that $\langle \phi_{n,\infty}, \phi_{\infty,n}\rangle$ is a projection from D_∞ to D_n.

Thus, because $\phi_{n,\infty}$ embeds every D_n isomorphically into (a subset of) D_∞, we have, modulo isomorphism, that $D_0 \subseteq D_1 \subseteq \ldots D_\infty$, and that D_∞ does, in fact, contain its own function space. (Intuitively, D_∞ contains all the D_n, and hence contains all the function spaces.) Thus we can identify each $d \in D_\infty$ with $\phi_{n,\infty}(d) \in D_\infty$ and treat d as though it were actually in D_∞. In this way, we can restate some of our earlier results; for instance, we have that for any $a \in D_\infty$, that $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \ldots$ and that $a = \sqcup\{a_0, a_1, \ldots\}$. In the latter case, a may be thought of as the limit of a sequence of better and better approximations to a.

We may now define application. Let $a, b \in D_\infty$, and define

$$a \cdot b = \sqsubseteq_{n\geq 0}\ \phi_{n,\infty}(a_{n+1}(b_n))$$

Intuitively, this means that $a \cdot b$ is the limit of a sequence of better and better approximations $a_{n+1}(b_n)$.

We may interpret variables, or terms that do not contain λ, as follows. Let $\rho : Vars \to D_\infty$ be an environment. Then define $[\![x]\!]_\rho = \rho(x)$ and $[\![PQ]\!]_\rho = [\![P]\!]_\rho \cdot [\![Q]\!]_\rho$.

Furthermore, this way of interpreting variables leads to an approximation in each D_n. To wit, $[\![x]\!]^n_\rho = (\rho(x))_n$ and $[\![PQ]\!]^n_\rho = [\![P]\!]^{n+1}_\rho([\![Q]\!]^n_\rho)$. Essentially, this approximation, for a variable x, is just the n-th member of the interpretation of x; for function application, the approximation is just a "stratified" version of what we would expect.

Rather than interpreting λ directly, it is easier to show that D_∞ is an extensional combinatory algebra, because all extensional combinatory algebras are λ-models. This leads to the somewhat unsatisfying situation where, given a λ-term, we cannot interpret it directly in D_∞, but must first instead pass through D_∞ as a combinatory algebra.

Earlier we remarked that we could define k_n as an approximation to $\mathbf{K}$ for each D_n. We can now define $k = \langle \bot_0, I_{D_0}, k_2, k_3, k_4, \ldots \rangle$. We have that for every $a, b \in D_\infty$, $k \cdot a \cdot b = a$. Likewise, we can define $s = \langle \bot_0, I_{D_0}, \psi_2(s_3), s_3, s_4, \ldots \rangle$. Having s and k, D_∞ is a combinatory algebra. With these definitions, we also have that D_∞ is extensional: that is, that if $a \cdot c = b \cdot c$ for every c, then $a = b$.

Finally, there are several interesting facts about D_∞ that we will simply state here. First, that the continuous functions from D_∞ to D_∞ are exactly the representable ones. Second, that $[D_\infty \to D_\infty]$ is a cpo that is isomorphic to D_∞. Our third property relates to the interesting notion of fixed points.

One of the advantageous of working with $\bot$ as an element representing non-termination in the way we have done is that it guarantees that λ-term interpretations have fixed points. In fact, given a cpo D and $\phi \in [D \to D]$, the least fixed point of ϕ is given by $\sqsubseteq_{n \geq 0} \phi^n(\bot)$. These least fixed points play a larger part in the theory of typed lambda calculus, where we must explicitly interpret a fixed-point finding operator, as such an operator is not definable in a typed system. This operator is definable in λ.

Finally, we will mention only that an equation $M = N$ is true in D_∞ iff the "infinite Bohm-trees" of M and N have the same "infinite normal form," providing a strong sense that D_∞ is a "natural" λ-model.

6.5 Connections to Category Theory

The construction of D_∞ can be accomplished inside the category *Cpo* using the categorical limit construction. Inside *Cpo*, if we regard arrows as projections, then we can find the limiting object from which every D_n can be projected from. First, we must define what a limit in a category is, and for that we must define a *cone*.

Definition 30. *Let F be a functor from category $\mathcal{J}$ to category $\mathcal{C}$. A cone of F is an object L in $\mathcal{C}$, together with a set of arrows $\{\phi_X : L \to F(X)\}$, one arrow for every object X of $\mathcal{J}$, such that for every arrow $f : X \to Y$ in $\mathcal{J}$, it is the case that $F(f) \circ \phi_X = \phi_Y$. That is, Figure 11 commutes.*

Intuitively, we have that whenever two objects X and Y are related by an arrow f in the source category $\mathcal{J}$, they are related in $\mathcal{C}$ under the functor F. The object L is a "source" in the target category $\mathcal{C}$ because it has arrows to X and Y under F. In this way, if we have some sequence of objects in $\mathcal{J}$ related by arrows, then the cone object L has arrows to the images of these objects in $\mathcal{C}$. For example, in a partial order, a lower bound is a cone. Viewed in this setting, the term "source" may be more appropriate than "cone." Remember,

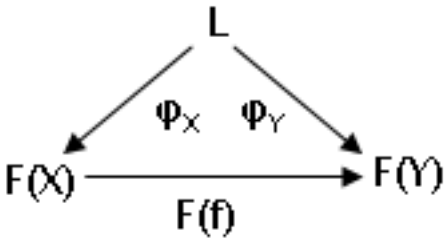

Figure 11: A Cone.

however, that a cone is defined relative to a functor. Finally, note that this construction is called a cone because the set of all these diagrams, being triangles, can be arranged to form a three-dimensional cone with L as the tip.

Simply put, the *limit* of a functor is a universal cone.

Definition 31. *Let F be a functor from $\mathcal{J}$ to $\mathcal{C}$ with a cone $\langle L, \phi_X \rangle$. This cone is a limit of F iff for any cone $\langle N, \psi_X \rangle$ of F, there exists exactly one arrow u from N to L such that $\phi \circ u = \psi_X$ for every object X. That is, the following diagram commutes.*

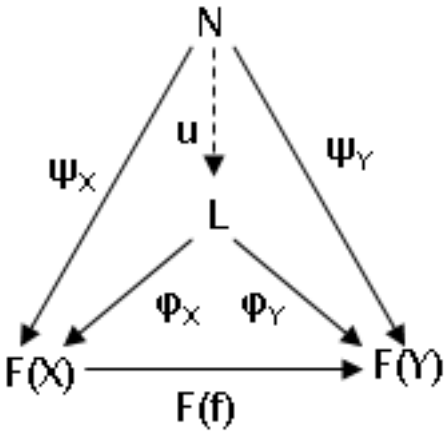

Figure 12: A Limit.

In other words, a limit captures the common essence of every cone in a unique way. In [5], a limit is described as an entity which has a privileged behavior among a class of objects that satisfy a certain property. Intuitively, the limit of a set of objects captures some property that each object approaches individually. In a partial order, a limit is a greatest lower bound; the greatest lower bound captures the property of being a lower bound in a way that intuitively feels like a limit.

The concept of a limit may appear strange because these limits seem to be "backwards," in the sense that we are only sure of a great many arrows leaving the limit; the only arrows we are sure of that point to the limit are those coming from cones. So in a sense, the limit is really the limit of the cones; that is, the object that all the cones point to: the limit is a

universal source. (One may be tempted to imagine a limit as a universal target. In fact, it matters very little if the limit is a source or target, because these notions are related through the categorical concept of duality. For the purposes of constructing D_∞, treating limits as sources is more useful.)

D_∞ is isomorphic to a special limit, $\varprojlim\langle D_n\phi_n\rangle$ in the cpo *Cpo*. Here, projections form the arrows (so the arrows are from D_{n+1} to D_n), and although it seems like we are taking the limit as $n \to \infty$, as our projections are from D_{n+1} to D_n, we are really speaking of the limit where all the projections can be projected from; that n increases as we take this limit is a consequence of how we numbered the D_n. In this setting, the term "source" for this limit is also quite appropriate than limit here, because D_∞ can be viewed as the source from which each D_n is projected. Ironically, this makes the concept of a categorical limit easier to grasp using the category D_∞ than with other, simpler categories. We may take this as an indication that D_∞ is easy to manipulate in a categorical setting.

More formally (but not completely formally), we can define a functor F that maps each cpo D to its function space: the cpo $[D \to D]$. To find D_∞, we are really trying to find a fixed point of the functor F. To do so we would like to let $n \to \infty$, but we are not sure in what sense to take the limit; that is, we can let n increase, but we do not know what the limit of a sequence of objects looks like; we do not know how to construct one. Moreover, we don't even know that such a limit exists in the category. Fortunately, we can formalize our above intuition of a "universal source" from which all D_n are projected from in terms of a category-theoretic construction called an *inverse limit*, denoted $\varprojlim$. In our case, the example of D_∞ as an inverse limit is actually more intuitive to grasp than both any other example and than the formal definition of what an inverse limit is, and so we will not give a formal definition here. Intuitively, an inverse limit exists when we have chains of objects (called inverse systems) for which there is an object that can serve as a universal source of projections, where projections are defined very similarly to how we defined them for D_∞.

One strength and weakness of category theory is that is is general enough that most category theoretic constructs can be defined in many different ways. In [4], for instance, there are three separate definitions of a ccc. Because of the great many techniques available in category theory, it can often be quite beneficial to treat D_∞ in this setting.

7 Conclusion

The path from the definition of a category to D_∞ is long and complex. Along the way, we have encountered applicative structures, which capture the essential properties of applicative elements that are so essential to λ-models. We have seen term models and syntax-free models: models built out of the syntax of λ itself, that seem trivial but point us in the right direction for reaching D_∞. We have seen glimpses of extremely abstract and powerful category-theoretic constructions like functors and cones. We have seen the connections of λ-models to category theory captured in the concepts of retraction, limit, and cartesian closure. When

we arrived at D_∞, we found both a classical approach to its construction and a categorical approach. We have witnessed computability characterized as topology.

It is these connections between fields, and the emergence of new fields from the constructions required to give meaning to such a powerful system as λ, that make studying lambda calculus so very rewarding.

References

[1] *Introduction to Combinators and λ-calculus*. J. Roger Hindley and Jonathan P. Seldin, Cambridge University Press, 1996.

[2] *Domains and Lambda-Calculi*. Roberto M. Amadio and Pierre-Louis Curien, Cambridge University Press, 1998.

[3] *The Lambda Calculus*. H.P. Barendregt, North-Holland Publishing Company, 1981.

[4] *Foundations for Programming Languages*. John C. Mitchell, MIT University Press, 1996.

[5] *Categories, Types, and Structures*. Andrea Asperti and Giuseppe Longo, MIT University Press, 1991.