Chamchuri Journal of Mathematics
Volume 6(2014), 57–88
http://www.math.sc.chula.ac.th/cjm

Survey Article

Models of the Lambda Calculus: An Introduction

Mark E. Hall

Received 21 February 2014
Revised 6 November 2014
Accepted 6 November 2014

Abstract: The λ-calculus is a symbolic formalism for describing and calculating with functions. To give meaning to expressions in the λ-calculus they must be interpreted in terms of standard mathematical objects such as sets and functions. Each such interpretation is called a model of the λ-calculus. Because the concept of function embodied in the λ-calculus is very general—for example, a λ-calculus function can be applied to itself—we cannot model λ-calculus functions as ordinary mathematical functions. Consequently such models require some new ideas that turn out to have applications in other areas of mathematics, and in computer science as well. This paper presents an introduction to models of the λ-calculus that is largely self-contained, although most technical details are omitted. It starts with a brief exposition of the λ-calculus itself, followed by some basic theory of λ-models in general and then descriptions of two specific models, including Dana Scott's D_∞, the first non-trivial model discovered. It concludes with suggestions for further reading.

Keywords: Lambda Calculus, Lambda Model, Applicative Structure, Complete Partial Order, Combinatorial Completeness

2000 Mathematics Subject Classification: 03B40, 03C65

1 Introduction

Surely every mathematician is at least acquainted with predicate logic. If nothing else, we all use it at times as a precise, unambiguous shorthand for writing mathematical statements, such as the condition for the function $f : \mathbb{R} \to \mathbb{R}$ to be continuous at the number a:

$$\forall \epsilon > 0 \; \exists \delta > 0 \; \forall x \in \mathbb{R} \big(|x - a| < \delta \Rightarrow \big| f(x) - f(a) \big| < \epsilon \big).$$

If we want to be rigorous, however, then we must recognize that the statements we write in predicate logic are just strings of symbols that have no inherent meaning. To give a statement a meaning it must be interpreted relative to a *structure*, which assigns a concrete mathematical meaning to each symbol in the statement. If the interpretation of the statement in the structure yields the value "true" then we call the structure a *model* of the statement. (The reader interested in the details of statements, structures, interpretations, and models in predicate logic can look in any of several good books on the subject, such as [10].)

Less familiar among our ranks is the λ-calculus, which is a formal notation for working with functions invented by Alonzo Church (see [5]). The basic idea of the λ-calculus is surprisingly simple: We use the notation $\lambda x.E$ to describe the function that maps x to E. For example, $\lambda x.x + 1$ denotes the function that maps x to $x + 1$ (i.e., the successor function on the natural numbers) and $\lambda x.x$ denotes the identity function that maps every object to itself. When the function denoted by $\lambda x.E$ is applied to an argument the resulting value can be calculated by substituting the argument for x in E. For instance, we calculate $(\lambda x.x+1)(3)$ by substituting 3 for x in $x+1$, obtaining $3+1$, which of course equals 4. Functions of multiple variables are handled via a trick known as *currying*, in which a function of one variable applied to the first argument has a value that is again a function and can be applied to the second argument, and so forth, until all of the arguments have been used. For example, the addition function can be denoted by $\lambda x.\lambda y.x+y$; applying it to the arguments 2 and 3 produces the expected final result of 5 as follows:

$$(\lambda x.\lambda y.x + y)(2)(3) = \big(\big(\lambda x.(\lambda y.x + y) \big)(2) \big)(3) = (\lambda y.2 + y)(3) = 2 + 3 = 5.$$

For a more extensive discussion of the intuition behind the λ-calculus, see Chapter I of Church's original book, [6].

The most interesting aspect of the λ-calculus is the fact that an expression in the λ-calculus, called a λ-*term*, can be applied to any λ-term—including itself. For example, if $T = \lambda x.xxx$ then $T(T) = (\lambda x.xxx)(T) = TTT$. It is this capability that gives the λ-calculus its power. For instance, it allows us to construct a fixed-point combinator, which is a λ-term Y with the property that for any λ-term f we have $f\big(Y(f)\big) = Y(f)$; that is, $Y(f)$ is a fixed point of f. The existence of a fixed-point combinator ensures that every λ-term f has a fixed point p; this in turn allows us to define functions recursively in the λ-calculus. As an example, let us look at how we can use fixed points to show the existence of a λ-term that calculates the factorial function. Consider the λ-term

$$T = \lambda F.\lambda x.\text{if } x = 0 \text{ then } 1 \text{ else } x \cdot F(x-1)$$

and let f be a fixed point of T, so that $T(f) = f$. Note that

$$\begin{aligned} T(f) &= \big(\lambda F.\lambda x.\text{if } x = 0 \text{ then } 1 \text{ else } x \cdot F(x-1)\big)(f) \\ &= \lambda x.\text{if } x = 0 \text{ then } 1 \text{ else } x \cdot f(x-1). \end{aligned}$$

Now the fact that $f = T(f)$ shows that for any n we have

$$\begin{aligned} f(n) = T(f)(n) &= \big(\lambda x.\text{if } x = 0 \text{ then } 1 \text{ else } x \cdot f(x-1)\big)(n) \\ &= \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot f(n-1), \end{aligned}$$

and hence f does indeed compute the factorial function.

Thus the λ-calculus formalizes a very general and powerful concept of function (some authors prefer the word "operation", to distinguish it from the standard mathematical notion of function). With such generality comes the potential for broad applicability, and consequently the influence of the λ-calculus is slowly growing, especially in computer science. This influence is most readily apparent in the area of functional programming languages, where it not only provides a theoretical foundation (see [20], Chapter 3) but is also a source of implementation ideas (see [19]). However, in the last several years even mainstream programming languages such as C# ([18]) and C++ ([8]) have come to support the use of lambda expressions to define simple, anonymous functions. The influence of the λ-calculus is not limited to computer science, however. It provided one of the earliest definitions of computable function and inspired much of the early work in recursion theory. Another example is Klaus Grue's Map Theory (see [12]), an alternative foundation for mathematics that takes mappings as its primitive

objects; it uses the λ-calculus as the formal language for stating its axioms and their consequences. For a much more extensive and detailed discussion of the influence of the λ-calculus see Barendregt's article, [2].

As with predicate logic, λ-terms are just strings of symbols that have no meaning. To give them meaning we must construct a model of the λ-calculus, which interprets all λ-terms as elements of some set and also describes what it means to apply one λ-term to another. Unfortunately, as will be explained in more detail later on, the fact that every λ-term can be applied to itself means we cannot model a λ-term with an ordinary function; we must be more clever than that. Thus we are faced with the challenging—and intriguing—problem of how to model objects that behave much like functions, yet cannot be represented by standard mathematical functions. It is no surprise that more than 30 years passed between the time Church invented the λ-calculus and Dana Scott and Christopher Strachey constructed its first non-trivial model (see [25], [21], and [22]). Nonetheless, the effort required to construct and understand λ-models is worthwhile, because of the insight they give us into the λ-calculus. Furthermore, some of the concepts introduced in these constructions have proven useful in other fields. For example, the continuous lattices in Scott and Strachey's original model became the basis for domain theory, which plays an important role in the theory of denotational semantics and related areas in the theory of programming languages. In fact, Scott and Strachey's model was born from their efforts to rigorously define the meanings of recursively-defined programs.

The goal of this article is to give the reader an introduction to how we can model the expanded concept of operation embodied in the λ-calculus using the sets and functions of everyday mathematics. It is organized as follows: Section 2 gives a brief, semi-rigorous introduction to the λ-calculus; the reader who is already acquainted with it can safely skip this section, since in the rest of the article we only make use of basic concepts and definitions and adhere to standard notation. Section 3 presents two equivalent definitions of a λ-model, plus a few general results about λ-models that will be useful when we construct some specific models, and are interesting in their own right. Section 4 briefly describes the term model, an essentially trivial model of the λ-calculus, then shows how to construct one of the simplest non-trivial λ-models: Engeler's model D_A. This is followed by an overview of the construction of Scott's model D_∞, which was the first non-trivial λ-model discovered, in Section 5. We finish up with a few suggestions for further reading in Section 6.

2 An Overview of the λ-Calculus

Since we do not assume the reader is familiar with the λ-calculus, an introduction to its basic theory is provided in this section. For those interested in more details, [14] (or the earlier version [13]) is an excellent introductory account, while [1] is a standard reference for the core theory, although a bit challenging to read.

2.1 λ-Terms

As noted in the Introduction, the λ-calculus is a notation for talking about functions (operations), which formalizes their two key concepts:

1. Defining a function, by giving a formula that tells how to compute the function's value for any given argument (called abstraction).
2. Applying a function to an argument (called application).

For abstraction, if E is an expression then $\lambda x.E$ denotes the function whose value is E when the argument is x, while (fa) denotes the application of the function f to the argument a. Precise definitions of these are as follows:

Note 2.1. There are in fact several varieties of λ-calculus; the one we will be defining and working with here is technically called the *pure, untyped λ-calculus*.

Definition 2.2. Let *Var* be a countably infinite set of symbols, called *variables*. An *expression* is a finite sequence of symbols each of which is either an element of *Var* or one of the symbols λ, ., (, or). The set of all *λ-terms*, Λ, is the smallest set of expressions satisfying the following properties:

(a) $Var \subseteq \Lambda$;

(b) if $M, N \in \Lambda$ then $(MN) \in \Lambda$; and

(c) if $M \in \Lambda$ and $x \in Var$ then $(\lambda x.M) \in \Lambda$.

A λ-term of the form (MN) is called an *application*, while a λ-term of the form $(\lambda x.M)$ is called an *abstraction*. Since we will not be talking about any other type of terms, from here on we will refer to λ-terms as just *terms*.

When writing terms we will make use of two notational shorthand devices to reduce the number of symbols we need to write. The first shorthand device

we will make use of is to omit parentheses whenever we can do so while still making the intended meaning clear. This includes adopting the convention of association to the left when interpreting sequences of unparenthesized applications, whereby we group the two leftmost terms together, then group this pair with the third term from the left, and so on. Thus, for example, $MNPQ$ means $\big(\big((MN)P\big)Q\big)$ and $xz(yz)$ means $\big((xz)(yz)\big)$. Another convention we will adopt is that when interpreting an unparenthesized abstraction we will take the term following the dot to be the longest term that is consistent with the syntax of terms and any parentheses that do exist. Hence, for example, $\lambda x.\lambda y.xz(yz)$ means $\lambda x.\big(\lambda y.\big(xz(yz)\big)\big)$.

The second shorthand device we will make use of is to combine the λ's in abstractions of abstractions into a single λ. For example, $\lambda x.\lambda y.\lambda z.xz(yz)$ will often be written as $\lambda xyz.xz(yz)$.

There are several notions of equality and equivalence in the λ-calculus, so to avoid confusion we use different symbols for each one. In particular, if M and N are terms then the notation $M \equiv N$ denotes that M and N are *syntactically equal*, which means that they are the exact same sequences of symbols.

A term will often contain several variables, and each variable may occur multiple times in the term. Note, however, that occurrences of the variable x in the term $M \equiv \lambda x.N$ have a special role, since they denote the value of the argument when the "function" M is applied to an argument. Thus, we say that all occurrences of x in the term $\lambda x.N$ are *bound*; any occurrence of a variable that is not bound is said to be *free*. A variable that occurs in a term is said to be a *free variable* of the term iff it has at least one free occurrence in the term. Note that a precise definition of free and bound occurrences is more complicated than the above suggests. The details may be found in [14], Definition 1.11.

As will be discussed in greater detail shortly, in order to calculate the result of applying a "function" $\lambda x.M$ to an argument N we must replace all free occurrences of x in M with N. The notation $[N/x]M$ will denote the term obtained by performing this replacement. Again, a precise definition of $[N/x]M$ is much more subtle and complex than the above description would suggest—in fact it is surprisingly complex—primarily because we must ensure that every free occurrence of a variable in N remains free in $[N/x]M$. Such precision is not needed for this exposition so we will not go into the details here; however, the reader is encouraged to look at them in [14], Definition 1.12, or a similar exposition.

Example 2.3. Let $M \equiv \lambda x.xy(\lambda y.yx) \equiv \lambda x.\big((xy)(\lambda y.yx)\big)$ and $N \equiv \lambda z.z$. Then the first occurrence of y in M is free but the second occurrence is bound, so

$$[N/y]M \equiv \lambda x.\big(x(\lambda z.z)(\lambda y.yx)\big).$$

We all recognize that the functions f and g defined on the real numbers by $f(x) = x^2$ and $g(y) = y^2$ are really the same function; the use of the two different variables x and y is irrelevant. Similarly, given an abstraction $\lambda x.N$, it is reasonable to expect that if we replace all free occurrences of x in N with some other variable y that is not free in N and change the λx to λy then the term obtained is essentially the same as the original term. In symbols, we are converting $M \equiv \lambda x.N$ to $\lambda y.[y/x]N$. This conversion is called a *change of bound variable*, or an *α-conversion*, in M. If the term P can be converted to the term Q by a finite (perhaps empty) sequence of changes of bound variables then we say that P is *α-equivalent* to Q, and write $P \equiv_\alpha Q$.

Example 2.4. Note that

$$\begin{aligned}\lambda xyz.xz(yz) \equiv \lambda x.\lambda y.\lambda z.xz(yz) &\equiv_\alpha \lambda x.\lambda y.\lambda u.[u/z]\big(xz(yz)\big)\\ &\equiv \lambda x.\lambda y.\lambda u.xu(yu)\\ &\equiv_\alpha \lambda v.[v/x]\big(\lambda y.\lambda u.xu(yu)\big)\\ &\equiv \lambda v.\lambda y.\lambda u.vu(yu) \equiv \lambda vyu.vu(yu).\end{aligned}$$

Thus $\lambda xyz.xz(yz) \equiv_\alpha \lambda vyu.vu(yu)$.

2.2 β-Reduction and β-Equality

The λ-calculus also has a notion of computation, called β-reduction. The idea is that since the term $(\lambda x.M)N$ denotes the "function" $\lambda x.M$ described above applied to the argument N, we can calculate the "value" of $(\lambda x.M)N$ by substituting N for all free occurrences of x in M. That is, $(\lambda x.M)N$ "evaluates" to $[N/x]M$. The term $(\lambda x.M)N$ is called a *β-redex* and the corresponding term $[N/x]M$ is called its *contractum*. If a term P contains a β-redex and Q is obtained from P by replacing that β-redex with its contractum then we say P *β-contracts* to Q. If the term P can be converted to the term Q via a finite (perhaps empty) sequence of β-contractions and α-conversions then we say P *β-reduces* to Q, and write $P \rhd_\beta Q$. Finally, if there is a sequence of terms $P \equiv P_0, P_1, \ldots, P_n \equiv Q$ such that

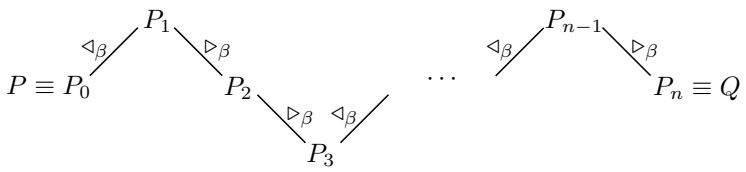

Figure 1: Illustration of $P =_\beta Q$

for each i either $P_i \triangleright_\beta P_{i+1}$ or $P_{i+1} \triangleright_\beta P_i$ then we say that P *β-equals* Q and write $P =_\beta Q$. Figure 1 illustrates $P =_\beta Q$.

Example 2.5. First a simple example. Let I be the term $\lambda x.x$ denoting the universal identity operation that was mentioned above. Then for any term M we have

$$\mathsf{I}M \equiv (\lambda x.x)M \triangleright_\beta [M/x]x \equiv M,$$

i.e., $\mathsf{I}M \triangleright_\beta M$, which also implies that $\mathsf{I}M =_\beta M$. Hence I has the behavior we expect of the identity operation.

Example 2.6. Now for a more complicated example. Consider the term

$$M \equiv (\lambda xy.y)\big((\lambda u.u)v\big)z \equiv \big((\lambda x.\lambda y.y)\big((\lambda u.u)v\big)\big)z.$$

The subterm $(\lambda x.\lambda y.y)\big((\lambda u.u)v\big)$ is a β-redex, which we can contract as follows:

$$M \equiv \big((\lambda x.\lambda y.y)\big((\lambda u.u)v\big)\big)z \triangleright_\beta [(\lambda u.u)v/x](\lambda y.y)z \equiv (\lambda y.y)z,$$

since x does not occur in $\lambda y.y$. On the other hand, the subterm $(\lambda u.u)v$ is also a β-redex, so we can contract it as well:

$$\begin{aligned} M \equiv \big((\lambda x.\lambda y.y)\big((\lambda u.u)v\big)\big)z \triangleright_\beta \big((\lambda x.\lambda y.y)([v/u]u)\big)z &\equiv \big((\lambda x.\lambda y.y)(v)\big)z \\ &\equiv (\lambda xy.y)vz. \end{aligned}$$

Hence we have that $M \triangleright_\beta (\lambda y.y)z$ and $M \triangleright_\beta (\lambda xy.y)vz$, which also means that $(\lambda y.y)z =_\beta (\lambda xy.y)vz$.

As the above example illustrates, a term can contain multiple β-redexes, and thus can β-reduce to different terms, depending on which β-redexes are contracted, and in what order. If we are thinking of β-reduction as computing,

this could be a problem, since it suggests the possibility that a single term could compute multiple different values. Fortunately, the following theorem, called the Church-Rosser Theorem, shows that while computations starting from the same term can yield different intermediate values, ultimately they must all produce the same value.

Theorem 2.7 ([14], Theorem 1.32; [1], Theorem 3.2.8(i))**.** *Let M be a term and suppose P and Q are terms such that $M \triangleright_\beta P$ and $M \triangleright_\beta Q$. Then there exists a term T such that $P \triangleright_\beta T$ and $Q \triangleright_\beta T$ (see Figure 2).*

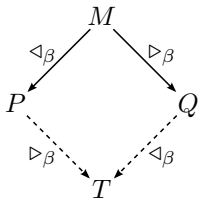

Figure 2: The Church-Rosser Theorem

As a corollary we get the following result, often called the Church-Rosser Theorem for β-equality, which says that if two terms are β-equal then they both compute the same value (note that the converse is obvious from the definition of β-equality).

Corollary 2.8 ([14], Theorem 1.41; [1], Theorem 3.2.8(ii))**.** *Suppose P and Q are terms such that $P =_\beta Q$. Then there exists a term T such that $P \triangleright_\beta T$ and $Q \triangleright_\beta T$.*

Because of this all terms that are β-equal to one another denote the same value. Therefore, when we construct models of the λ-calculus we must ensure that whenever two terms are β-equal they are interpreted in the same way in the model.

In practice, the Church-Rosser Theorem for β-equality is often used in its contrapositive form to prove that two terms are not β-equal. One special case is common enough to be worth mentioning here.

Corollary 2.9. *If M and N are terms that are not α-equivalent and neither M nor N contains a β-redex then $M \neq_\beta N$.*

As the above comments suggest, β-equality is important in the theory of λ-models. The following proposition lists some of its basic properties (see [1], Definition 2.1.4, Proposition 3.2.1, and Lemmas 3.2.4 and 3.2.7).

Proposition 2.10. *(a) β-Equality is an equivalence relation on Λ.*

(b) Suppose M, N, and P are terms, x is a variable, and $M =_\beta N$. Then $PM =_\beta PN$, $MP =_\beta NP$, $\lambda x.M =_\beta \lambda x.N$, and $[P/x]M =_\beta [P/x]N$.

3 General Theory of λ-Models

Before we look at any specific λ-models we obviously need a definition of what a λ-model is (although it should be noted that historically several specific models of the λ-calculus were constructed before any general definition of a λ-model was proposed). In fact, there are three equivalent definitions of a λ-model, each of which is useful in certain situations. In this section we will describe two of these definitions, showing how they are related to each other, and present a few basic results from the general theory of λ-models that will be needed in our constructions of specific λ-models. The third definition is more abstract, and although it is also useful it is not needed for this exposition, and has been omitted to help shorten its length.

Before starting the two definitions of a λ-model let us explain rigorously why it is not possible to model a λ-term M as an ordinary function $f_M : A \to B$ and application MN as applying f_M to f_N, i.e., $f_M(f_N)$. The problem arises when we consider the term MM, which would have to be modeled as $f_M(f_M)$, implying that f_M must be an element of A, the domain of f_M. In the standard set-theoretic definition of a function, this would mean there is a chain of membership relations, $f_M \in \cdots \in f_M$, which would allow us to construct an infinite descending chain of membership relations, $\cdots \in f_M \in \cdots \in f_M \in \cdots \in f_M$, violating the Axiom of Regularity in ZFC set theory (see [9] or [16]). Hence such an approach does not work. Dana Scott's model D_∞, described in Section 5 below, uses a clever trick to get around this problem.

3.1 First Definition: Interpretations

The first definition of a λ-model is similar to the definition of a model in first-order logic: We want to interpret terms as elements of some set (which, to help avoid

trivial models, is assumed to contain at least two elements), in such a way that if $M =_\beta N$ then M and N have the same interpretation. Specifically, we will have a set D and an interpretation function $[\![\ ]\!] : \Lambda \to D$ with the property that $M =_\beta N$ implies $[\![M]\!] = [\![N]\!]$. Now, many terms contain free variables, which can have arbitrary interpretations—after all, a free variable can be replaced by any term via substitution. Thus, when interpreting a term we must first assign values to its free variables, which is done via a function $\rho : \mathit{Var} \to D$, called a *valuation (of variables)*. Hence, we actually have a separate interpretation $[\![\ ]\!]_\rho : \Lambda \to D$ for each valuation ρ.

Notation 3.1. Let D be a set and $\rho : \mathit{Var} \to D$ a valuation. For each variable $x \in \mathit{Var}$ and element $d \in D$, $[d/x]\rho$ will denote the valuation $\sigma : \mathit{Var} \to D$ such that $\sigma(y) = \rho(y)$ for all variables $y \neq x$ and $\sigma(x) = d$.

As we noted above, we cannot interpret terms as ordinary functions and application as function application. The key to getting around this problem is to observe that application is a binary operation that combines two terms to produce another term. Thus, we want to have a binary operation $\bullet$ on D, so we can define $[\![MN]\!]_\rho$ to be $[\![M]\!]_\rho \bullet [\![N]\!]_\rho$. This also gives us a clue as to how we can interpret abstractions. Since $(\lambda x.M)N =_\beta [N/x]M$ we must have that

$$[\![\lambda x.M]\!]_\rho \bullet [\![N]\!]_\rho = [\![(\lambda x.M)N]\!]_\rho = [\![[N/x]M]\!]_\rho. \tag{1}$$

Furthermore, since $[N/x]M$ replaces each free occurrence of x in M with N, it is reasonable to expect that the interpretation of $[N/x]M$ should be the same as the interpretation of M under the valuation that assigns the interpretation of N to x; in symbols, $[\![[N/x]M]\!]_\rho = [\![M]\!]_{[d/x]\rho}$, where $d = [\![N]\!]_\rho$. Combining this with equation (1) yields the conclusion that $[\![\lambda x.M]\!]_\rho$ should satisfy the property that for all $d \in D$, $[\![\lambda x.M]\!]_\rho \bullet d = [\![M]\!]_{[d/x]\rho}$. Let us look at precise definitions now.

Definition 3.2. An *applicative structure* is a pair $\mathbb{D} = \langle D, \bullet \rangle$, where D is a set with at least two elements and $\bullet$ is a binary operation on D.

We do not assume the operation $\bullet$ in an applicative structure is associative, so to reduce the number of parentheses we need to write we adopt the convention of association to the left; for example, $a \bullet b \bullet c \bullet d$ means $\big((a \bullet b) \bullet c\big) \bullet d$.

Definition 3.3. A *λ-model* is a triple $\big\langle D, \bullet, [\![\ ]\!] \big\rangle$, where $\langle D, \bullet \rangle$ is an applicative structure and $[\![\ ]\!]$ is a function that assigns an element $[\![M]\!]_\rho$ of D to each valuation ρ and term M in such a way that the following properties are satisfied:

(a) For all variables x, $[\![x]\!]_\rho = \rho(x)$.

(b) For all terms M and N, $[\![MN]\!]_\rho = [\![M]\!]_\rho \bullet [\![N]\!]_\rho$.

(c) For all variables x, terms M, and elements $d \in D$, $[\![\lambda x.M]\!]_\rho \bullet d = [\![M]\!]_{[d/x]\rho}$.

(d) For all terms M and all valuations ρ and σ, $[\![M]\!]_\rho = [\![M]\!]_\sigma$ whenever $\rho(x) = \sigma(x)$ for all free variables x of M.

(e) For all terms M and all variables x and y, $[\![\lambda x.M]\!]_\rho = [\![\lambda y.[y/x]M]\!]_\rho$, provided that y is not a free variable of M.

(f) For all terms M and N, if for all $d \in D$ we have $[\![M]\!]_{[d/x]\rho} = [\![N]\!]_{[d/x]\rho}$ then $[\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$.

The above definition may seem long and cumbersome to the reader, but, from the discussion above, each of the properties (a)–(e) is one we would expect a reasonable λ-model to have. What about property (f)? First, observe that by property (c) the condition $\forall d \in D\big([\![M]\!]_{[d/x]\rho} = [\![N]\!]_{[d/x]\rho}\big)$ is equivalent to the condition $\forall d \in D\big([\![\lambda x.M]\!]_\rho \bullet d = [\![\lambda x.N]\!]_\rho \bullet d\big)$. Moreover, we are thinking of the abstraction $\lambda x.M$ as describing a function and $[\![\lambda x.M]\!]_\rho$ as its interpretation in D. We would expect this interpretation to act like a function on D, and indeed $[\![\lambda x.M]\!]_\rho \bullet d$ models the application of the interpretation of $\lambda x.M$ to the argument d. We can make this explicit by associating with each $a \in D$ a function $f_a : D \to D$, defined by $f_a(d) = a \bullet d$. In this new notation the condition $\forall d \in D\big([\![\lambda x.M]\!]_\rho \bullet d = [\![\lambda x.N]\!]_\rho \bullet d\big)$ becomes $\forall d \in D\big(f_{[\![\lambda x.M]\!]_\rho}(d) = f_{[\![\lambda x.N]\!]_\rho}(d)\big)$, which is equivalent to saying $f_{[\![\lambda x.M]\!]_\rho} = f_{[\![\lambda x.N]\!]_\rho}$, and property (f) becomes the statement that if $f_{[\![\lambda x.M]\!]_\rho} = f_{[\![\lambda x.N]\!]_\rho}$ then $[\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$. In other words, property (f) is saying that if the interpretations of two abstractions both induce the same function on D then in fact those interpretations should be the same. This is certainly a reasonable requirement, and it turns out it is needed to help prove that $[\![M]\!]_\rho = [\![N]\!]_\rho$ whenever $M =_\beta N$.

The concept of two elements of an applicative structure D inducing the same function on D will be needed again below, so let us give it a name and its own notation.

Definition 3.4. Let $\langle D, \bullet\rangle$ be an applicative structure and let $a, b \in D$. We say that a is *extensionally equivalent* to b iff $a \bullet d = b \bullet d$ for all $d \in D$ (i.e., iff $f_a = f_b$). We write $a \sim b$ to denote that a and b are extensionally equivalent.

It is easy to check that $\sim$ is an equivalence relation on D, and that using the new notation property (f) above is equivalent to saying that if $[\![\lambda x.M]\!]_\rho \sim [\![\lambda x.N]\!]_\rho$ then $[\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_\rho$.

The following theorem summarizes several fundamental properties of λ-models that can be proved from the above definition.

Theorem 3.5 ([14], Lemma 15.8, Lemma 15.10(a), and Theorem 15.12)**.** *Let $\langle D, \bullet, [\![\ ]\!]_\rho \rangle$ be a λ-model and let x, y be variables, M, N be terms and ρ, σ be valuations.*

(a) (Berry's extensionality property) If $[\![\lambda x.M]\!]_\rho \sim [\![\lambda y.N]\!]_\sigma$ then $[\![\lambda x.M]\!]_\rho = [\![\lambda y.N]\!]_\sigma$.

(b) $[\![[N/x]M]\!]_\rho = [\![M]\!]_{[d/x]\rho}$, where $d = [\![N]\!]_\rho$.

(c) If $M =_\beta N$ then $[\![M]\!]_\rho = [\![N]\!]_\rho$.

3.2 Second Definition: Syntax-Free λ-Models

The above definition is reasonably intuitive, especially for someone familiar with models in first-order logic. However, it is closely tied to the syntax of terms, and does not give much insight into what properties an applicative structure must have in order to be suitable for use in a λ-model. In this subsection we will give an alternative, equivalent definition that is phrased solely in terms of the applicative structure.

Let us start by assuming we have an applicative structure $\mathbb{D} = \langle D, \bullet \rangle$ and consider what additional properties it must satisfy in order for us to be able to define the interpretation $[\![M]\!]_\rho$ of every term M in such a way that we obtain a λ-model. It should be clear that the only difficult case is when $M \equiv \lambda x.N$ is an abstraction. Specifically, we need to determine how to define $[\![\lambda x.N]\!]_\rho$, assuming that we know $[\![N]\!]_\sigma$ for all valuations σ. From Definition 3.3, property (c), a crucial step is finding an element $a \in D$ such that

$$a \bullet d = [\![N]\!]_{[d/x]\rho} \quad \text{for all } d \in D. \tag{2}$$

Define the function $g_M : D \to D$ by $g_M(d) = [\![N]\!]_{[d/x]\rho}$. Then the crucial step amounts to finding an element $a \in D$ such that $g_M(d) = a \bullet d$ for all $d \in D$, which is often expressed by saying that g_M is *representable in* D. Since D contains at least two elements the cardinality of the set of all functions from D to D is greater

than the cardinality of D, so showing that g_M is representable in D is clearly going to be non-trivial. Let us first consider the special case that M is of the form $\lambda x_1 x_2 \cdots x_n.P$, where P consists only of variables combined using application.

A term P is called a *combination of variables* iff $P \equiv x$ for some variable x or $P \equiv (QR)$ for some combinations of variables Q and R. Let P be a combination of variables whose free variables all lie in the set $\{x_1, x_2, \ldots, x_n\}$. We can define a function $f_P : D^n \to D$ using induction on the length of P, by

$$f_P(d_1, d_2, \ldots, d_n) = \begin{cases} d_k & \text{if } P \equiv x_k, \\ f_Q(d_1, d_2, \ldots, d_n) \bullet f_R(d_1, d_2, \ldots, d_n) & \text{if } P \equiv (QR). \end{cases}$$

Intuitively, $f_P(d_1, d_2, \ldots, d_n)$ is a combination of the elements $d_1, d_2, \ldots, d_n$ (perhaps repeated) using $\bullet$.

Example 3.6. Let $P \equiv x(yx)yy$. Then P is a combination of variables with free variables x and y, and for all $d_1, d_2 \in D$ we have

$$f_P(d_1, d_2) = d_1 \bullet (d_2 \bullet d_1) \bullet d_2 \bullet d_2.$$

Suppose now that we are able to construct a λ-model using $\mathbb{D}$. Let P be a combination of variables whose free variables all lie in the set $\{x_1, x_2, \ldots, x_n\}$ and let $M \equiv \lambda x_1 x_2 \cdots x_n.P$. By repeated application of property (c) of Definition 3.3 we obtain that for any valuation ρ and any elements $d_1, d_2, \ldots, d_n \in D$,

$$\begin{aligned} [\![M]\!]_\rho \bullet d_1 \bullet d_2 \bullet \cdots \bullet d_n &= [\![\lambda x_1 x_2 \cdots x_n.P]\!]_\rho \bullet d_1 \bullet d_2 \bullet \cdots \bullet d_n \\ &= [\![P]\!]_{[d_n/x_n]\ldots[d_2/x_2][d_1/x_1]\rho}. \end{aligned}$$

Note that M has no free variables, so by property (d) of Definition 3.3 $[\![M]\!]_\rho$ is independent of ρ. Let $a_P = [\![M]\!]_\rho$; we now have that

$$a_P \bullet d_1 \bullet d_2 \bullet \cdots \bullet d_n = [\![P]\!]_{[d_n/x_n]\ldots[d_2/x_2][d_1/x_1]\rho}$$

for all $d_1, d_2, \ldots, d_n \in D$. It is easy to show that $[\![P]\!]_{[d_n/x_n]\ldots[d_2/x_2][d_1/x_1]\rho} = f_P(d_1, d_2, \ldots, d_n)$, so in fact we see that for any combination of variables P there is an element $a_P \in D$ such that $a_P \bullet d_1 \bullet d_2 \bullet \cdots \bullet d_n = f_P(d_1, d_2, \ldots, d_n)$ for all $d_1, d_2, \ldots, d_n \in D$.

Definition 3.7. An applicative structure $\langle D, \bullet \rangle$ is *combinatorially complete* iff for every combination of variables P there is an element $a_P \in D$ such that $a_P \bullet d_1 \bullet d_2 \bullet \cdots \bullet d_n = f_P(d_1, d_2, \ldots, d_n)$ for all $d_1, d_2, \ldots, d_n \in D$.

From the above discussion it is clear that an applicative structure $\mathbb{D}$ must be combinatorially complete in order to be used to construct a λ-model. The definition suggests it is difficult to verify that $\mathbb{D}$ is combinatorially complete. However, the following proposition shows that in fact it is sufficient to show that D contains two elements with certain properties.

Proposition 3.8 ([14], Theorem 14.29; [1], Theorem 5.1.10)**.** *An applicative structure $\langle D, \bullet \rangle$ is combinatorially complete iff there are elements $k, s \in D$ such that for all $a, b, c \in D$ we have $k \bullet a \bullet b = a$ and $s \bullet a \bullet b \bullet c = a \bullet c \bullet (b \bullet c)$.*

Let us now return to the problem of defining the interpretation $[\![M]\!]_\rho$ of a general abstraction $M \equiv \lambda x.N$. From Definition 3.3, property (c) it might appear that we can set $[\![M]\!]_\rho = a$ for any element $a \in D$ with property (2). However, in fact we must use a specific element a. Indeed, let y and z be variables that do not appear in N and let $\mathsf{E} \equiv \lambda yz.yz$. Then

$$\mathsf{E}M \equiv (\lambda yz.yz)(\lambda x.N) \rhd_\beta \lambda z.(\lambda x.N)z \rhd_\beta \lambda z.[z/x]N \equiv_\alpha \lambda x.N \equiv M,$$

so $\mathsf{E}M =_\beta M$. This implies that $[\![M]\!]_\rho = [\![\mathsf{E}M]\!]_\rho = [\![\mathsf{E}]\!]_\rho \bullet [\![M]\!]_\rho$, i.e., that $[\![M]\!]_\rho = e \bullet [\![M]\!]_\rho$, where $e = [\![\mathsf{E}]\!]_\rho$. Therefore, when we define $[\![M]\!]_\rho$ for an abstraction M we must find an element a of D that satisfies both property (2) and the condition $a = e \bullet a$. How can we find such an element a?

First, we note that although there are potentially many elements of D that satisfy property (2), they are all related. Indeed, if a and a' both satisfy property (2), then for all $d \in D$ we have that $a \bullet d = [\![N]\!]_{[d/x]\rho} = a' \bullet d$, which implies that $a \sim a'$. Similarly, if a satisfies property (2) and $a \sim a'$ then a' satisfies property (2) as well. Hence the set of all elements of D that satisfy property (2) forms an equivalence class under $\sim$. To define $[\![M]\!]_\rho$ we just need to pick out one element a in this equivalence class that satisfies $a = e \bullet a$.

Define a function $\Lambda : D \to D$ by $\Lambda(b) = e \bullet b$, and note that for any valuation ρ we have that

$$\Lambda(b) = [\![\mathsf{E}]\!]_\rho \bullet b = [\![\lambda yz.yz]\!]_\rho \bullet b = [\![\lambda z.yz]\!]_{[b/y]\rho}$$

Is $\Lambda(b) \sim b$? Let d be in D, and note that

$$\Lambda(b) \bullet d = [\![\lambda z.yz]\!]_{[b/y]\rho} \bullet d = [\![yz]\!]_{[d/z][b/y]\rho} = [\![y]\!]_{[d/z][b/y]\rho} \bullet [\![z]\!]_{[d/z][b/y]\rho} = b \bullet d.$$

Hence we do indeed have $\Lambda(b) \sim b$. In particular, if the element a satisfies property (2) then $\Lambda(a)$ also satisfies property (2). If we can show in addition that

$\Lambda(a) = e \bullet \Lambda(a)$ then given any element a that satisfies property (2), $\Lambda(a)$ will be a good candidate for the definition of $[\![M]\!]_\rho$.

Observe that $e \bullet \Lambda(a) = \Lambda\big(\Lambda(a)\big)$, so we would like to show that $\Lambda\big(\Lambda(a)\big) = \Lambda(a)$. In fact, we will show a stronger property, namely that if $a \sim b$ then $\Lambda(a) = \Lambda(b)$; the result $\Lambda\big(\Lambda(a)\big) = \Lambda(a)$ will then follow from the fact that $\Lambda(a) \sim a$. Assume therefore that $a \sim b$. Then $\Lambda(a) \sim a \sim b \sim \Lambda(b)$, so given a fixed valuation ρ we have that $[\![\lambda z.yz]\!]_{[a/y]\rho} \sim [\![\lambda z.yz]\!]_{[b/y]\rho}$. By part(a) of Theorem 3.5 this implies that $[\![\lambda z.yz]\!]_{[a/y]\rho} = [\![\lambda z.yz]\!]_{[b/y]\rho}$, i.e., $\Lambda(a) = \Lambda(b)$.

We now have a complete solution for how to define $[\![M]\!]_\rho$ when $M \equiv \lambda x.N$ is an abstraction. Indeed, given any element a of D that satisfies property (2), we know that $[\![M]\!]_\rho \sim a$. Furthermore, $[\![M]\!]_\rho = e \bullet [\![M]\!]_\rho$, so $[\![M]\!]_\rho = \Lambda\big([\![M]\!]_\rho\big) = \Lambda(a)$. That is, we must have $[\![M]\!]_\rho = \Lambda(a)$.

The reader has surely noticed that in fact our solution for how to define $[\![M]\!]_\rho$ is not truly complete, because we have not shown how to find an element a of D that satisfies property (2), or even that such an element must exist. It is possible to give a construction for an element a with the requisite property. However, the details are a bit complicated and do not yield deeper insight into the fundamental ideas of λ-models, so they have been omitted. The interested reader can find them in any complete exposition on λ-models, such as pages 238–239 of [14]. In any case, we now have enough background to motivate the following definition and theorem.

Definition 3.9. A *syntax-free λ-model* is a triple $\langle D, \bullet, \Lambda\rangle$ where $\mathbb{D} = \langle D, \bullet\rangle$ is an applicative structure, $\Lambda : D \to D$, and the following properties are satisfied:

(a) $\mathbb{D}$ is combinatorially complete;

(b) for all $a \in D$, $\Lambda(a) \sim a$;

(c) for all $a, b \in D$, if $a \sim b$ then $\Lambda(a) = \Lambda(b)$; and

(d) there exists an element $e \in D$ such that for all $a \in D$, $\Lambda(a) = e \bullet a$.

Theorem 3.10 ([14], Theorem 15.20)**.** *If $\langle D, \bullet, \Lambda\rangle$ is a syntax-free λ-model then we can construct a λ-model $\big\langle D, \bullet, [\![\ ]\!]\big\rangle$ by defining*

(a) $[\![x]\!]_\rho = \rho(x)$ if x is a variable;

(b) $[\![MN]\!]_\rho = [\![M]\!]_\rho \bullet [\![N]\!]_\rho$; and

(c) $[\![\lambda x.N]\!]_\rho = \Lambda(a)$, *where* a *is any element of* D *such that* $a \bullet d = [\![N]\!]_{[d/x]\rho}$ *for all* $d \in D$.

Conversely, if $\langle D, \bullet, [\![\ ]\!] \rangle$ *is a* λ*-model then we can construct a syntax-free* λ*-model* $\langle D, \bullet, \Lambda \rangle$ *by defining* $\Lambda(a) = e \bullet a$, *where* $e = [\![\lambda yz.yz]\!]_\rho$ *for any valuation* ρ.

Recall that Proposition 3.8 characterized combinatorial completeness in terms of the existence of two elements with certain properties. Since the function Λ in a syntax-free λ-model can be defined using the element e, this raises the question of whether the existence of Λ can be characterized in terms of the existence of an element satisfying the appropriate properties. Indeed it can.

Theorem 3.11 ([14], Discussion 15.23)**.** *Let* $\mathbb{D} = \langle D, \bullet \rangle$ *be an applicative structure such that* $\mathbb{D}$ *is combinatorially complete and there exists an element* $e \in D$ *such that*

(a) for all $a, b \in D$, $e \bullet a \bullet b = a \bullet b$; *and*

(b) for all $a, b \in D$, *if* $a \sim b$ *then* $e \bullet a = e \bullet b$.

Then $\langle D, \bullet, \Lambda \rangle$ *is a syntax-free* λ*-model, where* $\Lambda : D \to D$ *is defined by* $\Lambda(a) = e \bullet a$ *for all* $a \in D$.

A triple $\langle D, \bullet, e \rangle$ that satisfies the hypotheses of the preceding theorem is called a *loose Scott-Meyer* λ*-model* (see [23] and [17]). It follows from Proposition 3.8 that loose Scott-Meyer λ-models can be characterized completely in terms of the existence of elements of the applicative structure that satisfy certain properties, i.e., they can be characterized in completely algebraic terms. In particular, we now have a characterization of λ-models that is strictly internal to the applicative structure, which is sometimes the most convenient way to verify that a structure is indeed a λ-model.

There is one special case where defining Λ so as to obtain a syntax-free λ-model is trivial:

Theorem 3.12 ([14], Theorem 15.30)**.** *Suppose* $\langle D, \bullet \rangle$ *is a combinatorially complete applicative structure such that for all* $a, b \in D$, $a \sim b$ *implies* $a = b$. *Then defining* $\Lambda(a) = a$ *for all* $a \in D$ *makes* $\langle D, \bullet, \Lambda \rangle$ *a syntax-free* λ*-model.*

Proof. It is trivial that Λ satisfies parts (b) and (c) of Definition 3.9. For part (d), note that $P \equiv x_1$ is a combination of variables and for each $a \in D$, $f_P(a) = a$.

Hence by combinatorial completeness there exists an element $e \in D$ such that $e \bullet a = f_P(a) = a$ for all $a \in D$. That is, $\Lambda(a) = e \bullet a$ for all $a \in D$. □

An applicative structure $\langle D, \bullet \rangle$ satisfying the property that for all $a, b \in D$, $a \sim b$ implies $a = b$ is said to be *extensional*. Thus Theorem 3.12 can be stated succinctly as "Every extensional combinatorially complete applicative structure can be made into a syntax-free λ-model."

4 Engeler's Model D_A

Before we start on Engeler's model it should be noted that a very simple model, called the *term model*, can be constructed using the terms of the λ-calculus itself. In the term model the set D consists of all equivalence classes of λ-terms under β-equality, which is an equivalence relation by Proposition 2.10, part (a). For the binary operation $\bullet$ on D, define $[M]_\beta \bullet [N]_\beta = [MN]_\beta$, where $[M]_\beta$ denotes the equivalence class of M. It is straightforward to show that D contains elements k, s, and e with the requisite properties, and thus by Proposition 3.8 and Theorem 3.11 this yields a λ-model.

The construction of the term model is very simple, but because it is so closely tied to the syntax of the λ-calculus the model does not yield much insight. A much more useful model is D_A, which has the shortest known model construction, other than the term model. It was discovered by Erwin Engeler (see [11]), although Gordon Plotkin and Robert Meyer each had similar ideas several years earlier. It is constructed as a syntax-free λ-model, as follows.

Let A be a nonempty set. For any sets α and m let $(\alpha \mapsto m)$ denote the ordered pair $\big(A, (\alpha, m)\big)$, and note that

(i) By the Axiom of Regularity in ZFC set theory, $(\alpha \mapsto m) \notin A$.

(ii) For any sets α, β, m, and n, $(\alpha \mapsto m) = (\beta \mapsto n)$ iff $\alpha = \beta$ and $m = n$.

For any set X let $\mathbb{F}(X)$ denote the set of all finite subsets of X. Now, define an increasing sequence of sets $\big\{G_n(A)\big\}_{n=0}^{\infty}$ inductively by $G_0(A) = A$ and

$$G_{n+1}(A) = G_n(A) \cup \big\{(\alpha \mapsto m) \;\big|\; \alpha \in \mathbb{F}\big(G_n(A)\big) \text{ and } m \in G_n(A)\big\} \quad \text{for } n \geq 0.$$

Let $G(A) = \bigcup_{n=0}^{\infty} G_n(A)$ and note that $G(A)$ is the smallest set such that $A \subseteq G(A)$ and for every $\alpha \in \mathbb{F}\big(G(A)\big)$ and $m \in G(A)$, $(\alpha \mapsto m) \in G(A)$. Finally, let $D_A = \wp\big(G(A)\big)$, the set of all subsets of $G(A)$.

The applicative structure for this model is $\mathbb{D} = \langle D_A, \bullet \rangle$, where $a \bullet b$ is defined for all $a, b \in D_A$ by

$$a \bullet b = \big\{ m \in G(A) \;\big|\; \text{there exists } \beta \in \mathbb{F}(b) \text{ with } (\beta \mapsto m) \in a \big\}.$$

Define elements $k, s \in D_A$ by

$$k = \big\{ (\alpha \mapsto (\beta \mapsto m)) \;\big|\; \alpha, \beta \in \mathbb{F}\big(G(A)\big) \text{ and } m \in \alpha \big\},$$
$$s = \big\{ (\alpha \mapsto (\beta \mapsto (\gamma \mapsto m))) \;\big|\; \alpha, \beta, \gamma \in \mathbb{F}\big(G(A)\big) \text{ and } m \in \alpha \bullet \gamma \bullet (\beta \bullet \gamma) \big\}.$$

It is a straightforward calculation to show that for all $a, b, c \in D_A$ we have $k \bullet a \bullet b = a$ and $s \bullet a \bullet b \bullet c = a \bullet c \bullet (b \bullet c)$. Thus, by Proposition 3.8 $\mathbb{D}$ is combinatorially complete.

Intuitively, most elements of D_A act much like partial functions from $\mathbb{F}\big(G(A)\big)$ to $G(A)$, with $(\alpha \mapsto m) \in a$ denoting $a(\alpha) = m$. The following definitions are motivated by this intuition: For $a \in D_A$ let the *range of* a, denoted by $\mathrm{Rng}(a)$, be the set

$$\mathrm{Rng}(a) = \big\{ m \in G(A) \;\big|\; \text{there exists } \alpha \in \mathbb{F}\big(G(A)\big) \text{ with } (\alpha \mapsto m) \in a \big\}$$

and for $m \in G(A)$ let the *inverse image of* m *under* a, denoted by $a^{-1}(m)$, be the set

$$a^{-1}(m) = \big\{ \alpha \in \mathbb{F}\big(G(A)\big) \;\big|\; (\alpha \mapsto m) \in a \big\}.$$

Note that for any $m \in G(A)$, $m \in \mathrm{Rng}(a)$ iff $a^{-1}(m)$ is nonempty. For any nonempty collection $\mathcal{C}$ of finite subsets of $G(A)$ let $\mathrm{Min}(\mathcal{C})$ denote the set of all minimal sets (under the subset relation) in $\mathcal{C}$. We have the following characterization of the extensional equivalence relation $\sim$ on D_A:

Lemma 4.1. *For all* $a, b \in D_A$, $a \sim b$ *iff* $\mathrm{Rng}(a) = \mathrm{Rng}(b)$ *and for all* $m \in \mathrm{Rng}(a)$, $\mathrm{Min}\big(a^{-1}(m)\big) = \mathrm{Min}\big(b^{-1}(m)\big)$.

To finish our proof that D_A is a λ-model we need a function $\Lambda : D_A \to D_A$ that satisfies properties (b)–(d) of Definition 3.9. For $a \in D_A$ define

$$\Lambda(a) = \big\{ (\beta \mapsto m) \;\big|\; \beta \in \mathbb{F}\big(G(A)\big) \text{ and } m \in a \bullet \beta \big\}.$$

Combining this with the definition of $\bullet$ above we see that

$$\Lambda(a) = \big\{ (\beta \mapsto m) \;\big|\; \beta \in \mathbb{F}\big(G(A)\big) \text{ and } \beta \supseteq \alpha \text{ for some } (\alpha \mapsto m) \in a \big\}.$$

For $\alpha \in \mathbb{F}\big(G(A)\big)$ define the *upward cone* of α, denoted by $\mathrm{Up}(\alpha)$, by

$$\mathrm{Up}(\alpha) = \big\{\beta \in \mathbb{F}\big(G(A)\big) \;\big|\; \beta \supseteq \alpha\big\},$$

and extend this definition to an arbitrary collection $\mathcal{C}$ of sets in $\mathbb{F}\big(G(A)\big)$ by

$$\mathrm{Up}(\mathcal{C}) = \bigcup_{\alpha\in\mathcal{C}} \mathrm{Up}(\alpha).$$

It is easy to check that $\mathrm{Up}(\mathcal{C}) = \mathrm{Up}\big(\mathrm{Min}(C)\big)$. The proofs of the following basic results about $\Lambda(a)$ are straightforward:

Lemma 4.2. *For all $a \in D_A$ we have*

(a) $\Lambda(a) = \bigcup_{m\in\mathrm{Rng}(a)} \big\{(\beta \mapsto m) \;\big|\; \beta \in \mathrm{Up}\big(\mathrm{Min}(a^{-1}(m))\big)\big\}$;

(b) $\mathrm{Rng}\big(\Lambda(a)\big) = \mathrm{Rng}(a)$ *and for all* $m \in \mathrm{Rng}\big(\Lambda(a)\big)$, $\mathrm{Min}\big((\Lambda(a))^{-1}(m)\big) = \mathrm{Min}\big(a^{-1}(m)\big)$.

It follows immediately from Lemma 4.1 and part (a) of Lemma 4.2 that for all $a, b \in D_A$, if $a \sim b$ then $\Lambda(a) = \Lambda(b)$, and from Lemma 4.1 and part (b) of Lemma 4.2 that for all $a \in D_A$, $\Lambda(a) \sim a$. This shows that Λ satisfies properties (b) and (c) of Definition 3.9.

For property (d) of the definition we need to find an element $e \in D_A$ such that $\Lambda(a) = e \bullet a$ for all $a \in D_A$. Let

$$e = \big\{(\alpha \mapsto (\beta \mapsto m)) \;\big|\; \alpha, \beta \in \mathbb{F}\big(G(A)\big),\ m \in G(A),\ \text{and } m \in \alpha \bullet \beta\big\}.$$

Then for each $a \in D_A$ we have that

$$\begin{aligned}
e \bullet a &= \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \alpha \in \mathbb{F}(a),\ \beta \in \mathbb{F}\big(G(A)\big),\ \text{and } m \in G(A) \\
&\qquad\qquad \text{with } (\alpha \mapsto (\beta \mapsto m)) \in e\big\} \\
&= \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \alpha \in \mathbb{F}(a),\ \beta \in \mathbb{F}\big(G(A)\big),\ \text{and } m \in G(A) \\
&\qquad\qquad \text{with } m \in \alpha \bullet \beta\big\} \\
&= \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \alpha \in \mathbb{F}(a),\ \beta, \gamma \in \mathbb{F}\big(G(A)\big),\ \text{and } m \in G(A) \\
&\qquad\qquad \text{with } \gamma \subseteq \beta \text{ and } (\gamma \mapsto m) \in \alpha\big\} \\
&= \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \beta, \gamma \in \mathbb{F}\big(G(A)\big),\ \text{and } m \in G(A) \\
&\qquad\qquad \text{with } \gamma \subseteq \beta \text{ and } (\gamma \mapsto m) \in a\big\} \\
&= \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \gamma \in \mathbb{F}\big(G(A)\big),\ \text{and } m \in G(A)
\end{aligned}$$

$$\text{with } \beta \in \mathrm{Up}(\gamma) \text{ and } (\gamma \mapsto m) \in a\big\}$$
$$= \bigcup_{m \in \mathrm{Rng}(a)} \big\{(\beta \mapsto m) \;\big|\; \text{there exists } \gamma \in a^{-1}(m) \;\text{ with } \beta \in \mathrm{Up}(\gamma)\big\}$$
$$= \bigcup_{m \in \mathrm{Rng}(a)} \big\{(\beta \mapsto m) \;\big|\; \beta \in \mathrm{Up}\big(a^{-1}(m)\big)\big\}$$
$$= \bigcup_{m \in \mathrm{Rng}(a)} \big\{(\beta \mapsto m) \;\big|\; \beta \in \mathrm{Up}\big(\mathrm{Min}(a^{-1}(m))\big)\big\}$$
$$= \Lambda(a).$$

This completes the proof that D_A is a syntax-free λ-model.

5 Scott's Model D_∞

Dana Scott's[1] model D_∞ is important for several reasons: It was the first non-trivial model of the untyped λ-calculus, appearing at a time when many researchers in the field doubted it was possible to find an interpretation of all λ-terms in standard set theory. (In fact, a month before D_∞ was discovered Scott himself had argued strongly that finding such an interpretation was highly unlikely!) Furthermore, as noted in the Introduction, many of the concepts introduced in its construction have become standard tools in the semantics of programming languages.

The model presented below is a modified version of Scott's original model, which used complete lattices rather than the complete partial orders we will use. However, this is only a minor difference. Our exposition is based on [14], Chapter 16, which provides more detail, and in turn is derived from [1], Section 18.2.

5.1 Complete Partial Orders

Scott's key insight, explained in more detail below, was to "approximate" the object used to model a term using certain ordinary functions, then take a "limit" to get the object itself. The mathematical structure that is the basis for this process is the complete partial order, which we will now introduce. The reader interested in more details can consult sources such as [7].

[1] As stated in the Introduction, this model was the result of joint work by Dana Scott and Christopher Strachey; however, for the sake of brevity we will follow tradition and refer to it simply as "Scott's model".

Let $\langle D, \sqsubseteq \rangle$ be a poset (partially-ordered set). An element $b \in D$ is a *least element* iff $b \sqsubseteq d$ for all $d \in D$. Note that D may not have a least element, but if it does then the least element is unique, and will be denoted by $\bot$ (pronounced "bottom"). Let X be a subset of D. An *upper bound* of X is an element $u \in D$ such that $x \sqsubseteq u$ for all $x \in X$. A *least upper bound* of X is an element $\ell \in D$ such that ℓ is an upper bound of X and $\ell \sqsubseteq u$ for all upper bounds u of X. Again, a subset X of D may not have a least upper bound, but if it does then the least upper bound of X is unique and will be denoted by $\bigsqcup X$. A subset X of D is said to be *directed* iff X is nonempty and for every pair of elements $x, y \in X$ there exists an element $z \in X$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$.

Definition 5.1. A *complete partial order* (abbreviated as *cpo*) is a partial order $\langle D, \sqsubseteq \rangle$ such that

(a) D has a least element (denoted by $\bot$) and

(b) every directed subset X of D has a least upper bound (denoted by $\bigsqcup X$).

As is commonly done, we will normally write "the cpo D" rather than "the cpo $\langle D, \sqsubseteq \rangle$". Furthermore, we will adopt the convention that if D, D', and D'' are posets then the corresponding partial orderings will be denoted by $\sqsubseteq$, $\sqsubseteq'$, and $\sqsubseteq''$, respectively.

Example 5.2. Fix an object $\bot \notin \mathbb{N}$ and let $\mathbb{N}^+ = \mathbb{N} \cup \{\bot\}$. Define an ordering on $\mathbb{N}^+$ by

$$a \sqsubseteq b \quad \text{iff} \quad (a = \bot \text{ and } b \in \mathbb{N}) \text{ or } a = b$$

(see Figure 3). It is easy to check that the pair $\langle \mathbb{N}^+, \sqsubseteq \rangle$ is a cpo; it will be denoted simply by $\mathbb{N}^+$.

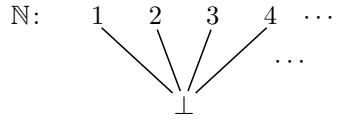

Figure 3: $\mathbb{N}^+$

The construction of D_∞ starts with $\mathbb{N}^+$. To take the next step we need to know how to construct cpos out of certain sets of functions on cpos.

5.2 Continuous Functions on Cpos

One important lesson from category theory is that any time we define a new kind of mathematical object we should also determine what the corresponding morphisms (mappings between objects of that kind) are. The basic rule is that the morphisms should preserve the structure of the objects they are mapping between. For the category of cpos the morphisms are the continuous functions defined below.

Definition 5.3. Let D and D' be cpos and $\phi : D \to D'$ a function.

(a) We say that ϕ is *monotonic* iff $a \sqsubseteq b$ implies $\phi(a) \sqsubseteq' \phi(b)$.

(b) We say that ϕ is *continuous* iff for all directed subsets X of D,

$$\phi\big(\bigsqcup X\big) = \bigsqcup \phi(X). \tag{3}$$

We use the notation $[D \to D']$ to denote the set of all continuous functions from D to D'.

Note that equation (3) means both that $\phi(X)$ has a least upper bound (there is no guarantee it is a directed set) and that the least upper bound equals $\phi\big(\bigsqcup X\big)$.

As the name suggests, it is also possible to give a topological definition of a continuous function from one cpo to another. A topology called the *Scott topology* can be defined on an arbitrary cpo, and a function is continuous in the sense above iff it is continuous with respect to the Scott topologies on D and D'.

It is a pair of easy exercises to prove that every continuous function is monotonic and the composition of continuous functions is continuous.

For cpos D and D' we can define a relation $\preceq$ on $[D \to D']$ by

$$\phi \preceq \psi \quad \text{iff} \quad \phi(d) \sqsubseteq' \psi(d) \text{ for all } d \in D. \tag{4}$$

It is easy to check that $\preceq$ is a partial order on $[D \to D']$. Furthermore, $[D \to D']$ has a least element, defined by

$$\bot(d) = \bot' \quad \text{for all } d \in D. \tag{5}$$

Finally, if Φ is a directed subset of $[D \to D']$ then for each $d \in D$ the set $\big\{\phi(d) \;\big|\; \phi \in \Phi\big\}$ is a directed subset of D' so we can define a function $\psi : D \to D'$ by

$$\psi(d) = \bigsqcup\big\{\phi(d) \;\big|\; \phi \in \Phi\big\} \quad \text{for all } d \in D. \tag{6}$$

It is straightforward to show that ψ is continuous and is the least upper bound of Φ. Hence we have the following result.

Proposition 5.4 ([14], Lemma 16.18)**.** *If D and D' are cpos then $[D \to D']$ is also a cpo under the partial ordering defined by (4). Its least element is given by (5) and for any directed subset Φ of $[D \to D']$, $\bigsqcup \Phi$ is the function ψ defined by (6).*

In particular, given a cpo D_0 we can construct a sequence $\{D_n\}_{n=0}^{\infty}$ of cpos inductively by defining $D_{n+1} = [D_n \to D_n]$ for all $n \geq 0$. Scott's model D_∞ uses such a sequence starting with $D_0 = \mathbb{N}^+$, but the results that follow are valid regardless of the choice of D_0. Now that we have the sequence of cpos $\{D_n\}_{n=0}^{\infty}$ let us look at how they are related to one another.

5.3 Projections of Cpos

Given two cpos D and D' it would seem natural to consider whether D can be embedded in D' (or vice versa), by which we would mean that there is a continuous monomorphism from D into D'. For our purposes we need a stronger relationship between D and D'.

Definition 5.5. Let D and D' be cpos. A *projection from D' to D* is a pair $\langle \phi, \psi \rangle$ of functions with $\phi \in [D \to D']$ and $\psi \in [D' \to D]$, such that

$$\psi \circ \phi = I_D \quad \text{and} \quad \phi \circ \psi \preceq I_{D'} \tag{7}$$

(here I_D denotes the identity function on D and similarly for $I_{D'}$).

It is easy to show that if $\langle \phi, \psi \rangle$ is a projection from D' to D then ϕ embeds D into D'. However, the additional existence of ψ with the properties in (7) turns out to be quite useful, as we shall see.

First, let us note that for the sequence of cpos $\{D_n\}_{n=0}^{\infty}$ constructed above we can define a projection $\langle \phi_n, \psi_n \rangle$ from D_{n+1} to D_n for each n, starting with a projection from D_1 to D_0, as follows. For each $d \in D_0$, let κ_d denote the constant function $\kappa_d(c) = d$ for all $c \in D_0$. It is easy to see that κ_d is continuous, so $\kappa_d \in [D_0 \to D_0] = D_1$. Now define $\phi_0 : D_0 \to D_1$ and $\psi_0 : D_1 \to D_0$ by $\phi_0(d) = \kappa_d$ for $d \in D_0$ and $\psi_0(c) = c(\bot_0)$ for $c \in D_1$ (where $\bot_0$ is the least element of D_0). It is straightforward to check that ϕ_0 and ψ_0 are continuous, $\psi_0 \circ \phi_0 = I_{D_0}$, and $\phi_0 \circ \psi_0 \preceq I_{D_1}$, so that $\langle \phi_0, \psi_0 \rangle$ is indeed a projection from D_1 to D_0. Now we define $\phi_n : D_n \to D_{n+1}$ and $\psi_n : D_{n+1} \to D_n$ inductively for $n > 0$ by

$$\phi_n(\sigma) = \phi_{n-1} \circ \sigma \circ \psi_{n-1} \quad \text{and} \quad \psi_n(\tau) = \psi_{n-1} \circ \tau \circ \phi_{n-1}$$

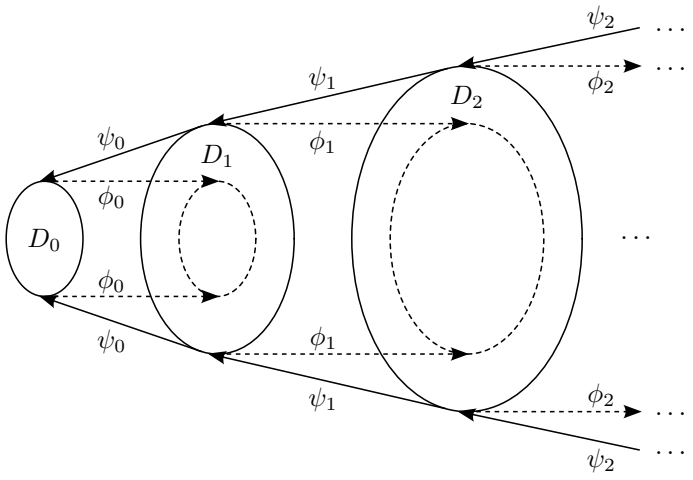

Figure 4: The sequence $D_0, D_1, D_2, \ldots$

for $\sigma \in D_n$ and $\tau \in D_{n+1}$. It takes some work, but is not difficult, to show that $\phi_n \in [D_n \to D_{n+1}]$, $\psi_n \in [D_{n+1} \to D_n]$, $\psi_n \circ \phi_n = I_{D_n}$, and $\phi_n \circ \psi_n \preceq I_{D_{n+1}}$. In other words, $\langle \phi_n, \psi_n \rangle$ is a projection from D_{n+1} to D_n. The result is the picture shown in Figure 4.

The functions ϕ_n and ψ_n only take us from one level of the sequence $\{D_n\}_{n=0}^{\infty}$ to an adjacent level. However, by composing them in the appropriate ways we can obtain functions that skip many levels.

Definition 5.6. For any $m, n \geq 0$ define $\phi_{m,n} : D_m \to D_n$ by

$$\phi_{m,n} = \begin{cases} \phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_{m+1} \circ \phi_m & \text{if } m < n, \\ I_{D_n} & \text{if } m = n, \\ \psi_n \circ \psi_{n+1} \circ \cdots \circ \psi_{m-2} \circ \psi_{m-1} & \text{if } m > n. \end{cases}$$

Since we already know that $\langle \phi_n, \psi_n \rangle$ is a projection from D_{n+1} to D_n it is easy to prove the following lemma:

Lemma 5.7 ([14], Lemma 16.33)**.** *Let $m, n \geq 0$. Then*

(a) $\phi_{m,n} \in [D_m \to D_n]$;

(b) if $m \leq n$ then $\phi_{n,m} \circ \phi_{m,n} = I_{D_m}$;

(c) if $m > n$ *then* $\phi_{n,m} \circ \phi_{m,n} \preceq I_{D_m}$ *;*

(d) if $m < n$ *then* $\langle \phi_{m,n}, \phi_{n,m} \rangle$ *is a projection from* D_n *to* D_m *;*

(e) if k *is between* m *and* n *then* $\phi_{k,n} \circ \phi_{m,k} = \phi_{m,n}$ *.*

5.4 The Construction of D_∞

We are now ready to define D_∞, the applicative structure used in Scott's model. As Figure 4 illustrates, the sequence $\{D_n\}_{n=0}^{\infty}$ is much like an increasing chain of sets, with each set nicely embedded inside the next one. Intuitively, we ought to be able to take the "union" of the sequence to obtain a set D_∞ such that each D_n embeds nicely inside D_∞. Although we will not define D_∞ as a union, this is a good intuition to use when thinking about it and its relationship to the D_n's.

Definition 5.8. Let D_∞ denote the set of all infinite sequences of the form

$$d = \langle d_0, d_1, d_2, \ldots \rangle$$

such that for all $n \geq 0$ we have $d_n \in D_n$ and $\psi_n(d_{n+1}) = d_n$. Define a relation $\sqsubseteq$ on D_∞ by

$$\langle d_0, d_1, d_2, \ldots \rangle \sqsubseteq \langle d'_0, d'_1, d'_2, \ldots \rangle \quad \text{iff} \quad d_n \sqsubseteq d'_n \text{ for all } n \geq 0.$$

Notation 5.9. To save some writing, we will adopt the convention that a subscript of n on an element of D_∞ will always denote the n^{th} term of that element's sequence. Thus, for example, if a, b are in D_∞ then a_n will denote the n^{th} term of the sequence that is a and $(a \bullet b)_n$ will denote the n^{th} term of the sequence that is $a \bullet b$. Also, if X is a subset of D_∞ then X_n will denote the set $\{\, a_n \mid a \in X \,\}$.

As expected, D_∞ is a cpo:

Proposition 5.10 ([14], Lemma 16.36)**.** *The pair* $\langle D_\infty, \sqsubseteq \rangle$ *defined above is a cpo, with least element*

$$\bot = \langle \bot_0, \bot_1, \bot_2, \ldots \rangle,$$

where $\bot_n$ *is the least element of* D_n*, and least upper bound of a directed subset* X *of* D_∞ *given by*

$$\bigsqcup X = \langle \bigsqcup X_0, \bigsqcup X_1, \bigsqcup X_2, \ldots \rangle.$$

It should not come as a surprise that for each $n \geq 0$ we have a pair of continuous functions that forms a projection from D_∞ to D_n. It is defined as follows:

Definition 5.11. For each $n \geq 0$ define $\phi_{n,\infty} : D_n \to D_\infty$ and $\phi_{\infty,n} : D_\infty \to D_n$ by

$$\phi_{n,\infty}(d) = \langle \phi_{n,0}(d), \phi_{n,1}(d), \phi_{n,2}(d), \ldots \rangle$$

for all $d \in D_n$ and

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

for all $d \in D_\infty$.

We have the following:

Proposition 5.12 ([14], Lemmas 16.38, 16.39, and 16.42)**.** *Let $m, n \geq 0$ with $m \leq n$ and $a, b \in D_\infty$.*

(a) The pair $\langle \phi_{n,\infty}, \phi_{\infty,n} \rangle$ is a projection from D_∞ to D_n.

(b) $\phi_{m,n}(a_m) \sqsubseteq a_n$

(c) $\phi_{m,\infty}(a_m) \sqsubseteq \phi_{n,\infty}(a_n)$

(d) $a = \bigsqcup_{n \geq 0} \phi_{n,\infty}(a_n)$

(e) $\phi_{n,\infty}\big(a_{n+1}(b_n)\big) \sqsubseteq \phi_{n+1,\infty}\big(a_{n+2}(b_{n+1})\big)$

Parts (c) and (d) of Proposition 5.12 are particularly interesting, because they suggest that intuitively we can regard the terms a_n of an element $a \in D_\infty$ as a sequence of increasingly good approximations to a, and a as the limit of the increasing sequence $\{a_n\}_{n=0}^{\infty}$. This is similar to the case of a function $f : A \to B$ where we have an increasing sequence of subsets $A_0 \subseteq A_1 \subseteq A_2 \subseteq \cdots$ of A whose union is A, $\bigcup_{n=0}^{\infty} A_n = A$. In that case we can think of the restrictions $f|_{A_0}, f|_{A_1}, f|_{A_2}, \ldots$ as a sequence of increasingly good approximations to f.

Furthermore, combining part (d) with part (e) suggests a way to define the binary operation $\bullet$ on D_∞. Suppose a, b are in D_∞. For each n, a_n and b_n are approximations of a and b, respectively. We cannot apply a_n to b_n, but we can apply a_{n+1} to b_n, since $a_{n+1} \in D_{n+1} = [D_n \to D_n]$. Thus, in a sense $a_{n+1}(b_n)$ is an approximation of a applied to b (more precisely, $\phi_{n,\infty}\big(a_{n+1}(b_n)\big)$ is the approximation, since a applied to b should yield an element of D_∞). By part (e) of Proposition 5.12, these approximations form an increasing sequence, so the approximations are getting better as n increases. Thus, it is reasonable to expect that a applied to b should be the limit of these approximations, i.e., their least upper bound. This is exactly how we define $\bullet$ on D_∞.

Definition 5.13. For all $a, b \in D_\infty$ we define

$$a \bullet b = \bigsqcup \{ \phi_{n,\infty}\big(a_{n+1}(b_n)\big) \mid n \geq 0 \}. \tag{8}$$

Note that by Proposition 5.12, part (e), the set $\{ \phi_{n,\infty}\big(a_{n+1}(b_n)\big) \mid n \geq 0 \}$ is linearly ordered, and thus a directed set, so the least upper bound in equation (8) always exists. Also, we see from this definition how Scott was able to cleverly sidestep the problem of self-application: While $a_n(a_n)$ is not defined, there is no problem with $a_{n+1}(a_n)$.

5.5 D_∞ is a λ-Model

We now have an applicative structure $\langle D_\infty, \bullet \rangle$; it only remains to show that it is a λ-model, which will be done by applying Proposition 3.8 and Theorem 3.12. The technical details of the proof are rather lengthy, so we will limit ourselves to defining the elements k and s needed for Proposition 3.8 and outlining a proof that $\langle D_\infty, \bullet \rangle$ is extensional.

Definition 5.14. (a) Let $n \geq 2$. For $a \in D_{n-1}$ define $\kappa_a : D_{n-2} \to D_{n-2}$ to be the constant function $\kappa_a(b) = \psi_{n-2}(a)$ for all $b \in D_{n-2}$. Then define $k_n : D_{n-1} \to D_{n-1}$ by $k_n(a) = \kappa_a$ for all $a \in D_{n-1}$.

(b) Let $n \geq 3$. For $a \in D_{n-1}$ and $b \in D_{n-2}$ define $\tau_{a,b} : D_{n-3} \to D_{n-3}$ by $\tau_{a,b}(c) = a\big(\phi_{n-3}(c)\big)\big(b(c)\big)$ for all $c \in D_{n-3}$, then define $\sigma_a : D_{n-2} \to D_{n-2}$ by $\sigma_a(b) = \tau_{a,b}$ for all $b \in D_{n-2}$. Finally, define $s_n : D_{n-1} \to D_{n-1}$ by $s_n(a) = \sigma_a$ for all $a \in D_{n-1}$.

Note that in order to show everything in the above definition is well-defined we must prove that κ_a, $\tau_{a,b}$ and σ_a are all continuous. The first two are trivial to check, while the last one requires a little more work, but is not hard. The functions k_n and s_n have the following properties:

Lemma 5.15 ([14], Exercise 16.30 and Note 16.31)**.** *(a) For all $n \geq 2$ we have that $k_n \in D_n$ and $\psi_n(k_{n+1}) = k_n$. In addition, $\psi_1(k_2) = I_{D_0} \in D_1$.*

(b) For all $n \geq 3$ we have that $s_n \in D_n$ and $\psi_n(s_{n+1}) = s_n$. In addition, $\psi_1\big(\psi_2(s_3)\big) = I_{D_0} \in D_1$.

Now we can define k and s in D_∞.

Definition 5.16. Let k and s be the following sequences:

$$k = \langle \bot_0, I_{D_0}, k_2, k_3, \ldots \rangle \quad \text{and} \quad s = \big\langle \bot_0, I_{D_0}, \psi_2(s_3), s_3, s_4, \ldots \big\rangle.$$

These are exactly what we need:

Lemma 5.17 ([14], Lemmas 16.51 and 16.53)**.** *The sequences k and s just defined are both elements of D_∞. Furthermore, for all $a, b, c \in D_\infty$, $k \bullet a \bullet b = a$ and $s \bullet a \bullet b \bullet c = a \bullet c \bullet (b \bullet c)$.*

Hence, by Proposition 3.8, $\langle D_\infty, \bullet \rangle$ is combinatorially complete. It only remains to show that $\langle D_\infty, \bullet \rangle$ is extensional. The idea for the proof is as follows: Let a and b be elements of D_∞ such that $a \sim b$, i.e., $a \bullet c = b \bullet c$ for all $c \in D_\infty$. Fix $m \geq 0$ and let d be an arbitrary element of D_m. Let $c = \phi_{m,\infty}(d)$. It can be shown (see the proof of Theorem 16.54 in [14]) that $(a \bullet c)_m = a_{m+1}(d)$ and $(b \bullet c)_m = b_{m+1}(d)$. Hence $a_{m+1}(d) = (a \bullet c)_m = (b \bullet c)_m = b_{m+1}(d)$. Therefore $a_{m+1} = b_{m+1}$. In other words, $a_n = b_n$ for all $n > 0$. But then $a_0 = \psi_0(a_1) = \psi_0(b_1) = b_0$ as well. Consequently $a = b$. Now by Theorem 3.12 $\langle D_\infty, \bullet \rangle$ is a syntax-free λ-model.

In addition to being a λ-model, D_∞ has some interesting properties as a cpo. For example, a function $\phi : D_\infty \to D_\infty$ is continuous iff $\phi = f_d$ for some $d \in D_\infty$ (recall that f_d is defined by $f_d(a) = d \bullet a$ for all $a \in D_\infty$). This, together with the fact that $\langle D_\infty, \bullet \rangle$ is extensional, implies that the function $\Phi : D_\infty \to [D_\infty \to D_\infty]$ defined by $\Phi(d) = f_d$ is a bijection. In fact, an even stronger fact can be proved, namely that Φ is an isomorphism of cpos, so $D_\infty \cong [D_\infty \to D_\infty]$. See [1], Theorems 18.2.15 and 18.2.16, for the details.

Before leaving D_∞ it should be noted that although our construction started with $D_0 = \mathbb{N}^+$, the only property of $\mathbb{N}^+$ we used was that it is a cpo. Therefore, replacing $\mathbb{N}^+$ with any other cpo as D_0 also produces a λ-model, which may have additional interesting properties, depending on the choice of D_0.

6 Further Reading

The above exposition has clearly omitted most details, and has only scratched the surface of the theory of λ-models, let alone the theory of the λ-calculus. So where should the reader interested in learning more go from here? The first step would be to obtain a solid grounding in the λ-calculus itself, since the introduction provided

in Section 2 above is only a sketch of the bare minimum needed to talk about λ-models. As suggested at the end of that section, Hindley and Seldin's book, [14] (or its predecessor, [13]), is an excellent place to start, covering all of the basic theory with a minimum of prerequisites. The reader who is interested in a serious study of λ-models may also want to dig into Barendregt's book, [1], to become familiar with more advanced aspects of the theory. Where to go from there depends on the reader's interests. Hindley and Seldin's book has an extensive bibliography, as does Chantal Berline's article, [3], which also presents a more systematic and algebraic approach to constructing and classifying λ-models. Finally, at the end of Dana Scott's notes derived from the slides of a few of his recent conference presentations, [24], is a very wide-ranging bibliography of books illustrating some of the interplay among the λ-calculus, logic, recursive function theory, category theory, and programming-language semantics. The three of these should provide the reader with plenty of interesting suggestions for further reading.

Acknowledgments: The author would like to thank the two anonymous reviewers for numerous helpful suggestions.

References

[1] H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, (revised ed.), North-Holland, 1984.

[2] H.P. Barendregt, The impact of the lambda calculus in logic and computer science. *Bulletin of Symbolic Logic*, **3**(1997), 181–215.

[3] C. Berline, From computation to foundations via functions and application: the λ-calculus and its webbed models. *Theoretical Computer Science*, **249**(2000), 81–161.

[4] C. Berline and K. Grue, A kappa-denotational semantics for Map Theory, in ZFC+SI. *Theoretical Computer Science*, **179**(1997), 137–202.

[5] A. Church, An unsolvable problem of elementary number theory. *American Journal of Mathematics*, **58**(1936), 354–363.

[6] A. Church, *The Calculi of Lambda-Conversion*, Princeton University Press, 1941.

[7] B.A. Davey and H.A. Priestley, *Introduction to Lattices and Order*, (2nd ed.), Cambridge University Press, 2002.

[8] S. Du Tois, *Working Draft, Standard for Programming Language C++*, International Organization for Standardization, 2012. Available online at `http://isocpp.org/std/the-standard`

[9] H.B. Enderton, *Elements of Set Theory*, Academic Press, 1977.

[10] H.B. Enderton, *A Mathematical Introduction to Logic*, (2nd ed.), Academic Press, 2001.

[11] E. Engeler, Algebras and combinators. *Algebra Universalis*, **13**(1981), 389–392.

[12] K. Grue, Map Theory. *Theoretical Computer Science*, **102**(1992), 1–133.

[13] J.R. Hindley and J.P. Seldin, *Introduction to Combinators and λ-Calculus*, Cambridge University Press, 1986.

[14] J.R. Hindley and J.P. Seldin, *Lambda-Calculus and Combinators, an Introduction*, Cambridge University Press, 2008. (This is an updated and rewritten version of [13].)

[15] J.L. Krivine, *Lambda-Calculus, Types, and Models* (originally published in French as *Lambda-calcul, types, et modèles*), Masson, 1993.

[16] K. Kunen, *Set Theory: An Introduction to Independence Proofs*, Elsevier, 1980.

[17] A.R. Meyer, What is a model of the lambda calculus? *Information and Control*, **52**(1982), 87–122. (The journal is now named *Information and Computation*.)

[18] Microsoft Corporation, Lambda Expressions (C# Programming Guide). [Online] Available: `http://msdn.microsoft.com/en-us/library/bb397687.aspx`, (2014).

[19] S.L. Peyton Jones, *The Implementation of Functional Programming Languages*, Prentice Hall, 1987.

[20] R. Plasmeijer and M. van Eekelen, *Functional Programming and Parallel Graph Rewriting*, Addison-Wesley, 1993.

[21] D.S. Scott, Outline of a mathematical theory of computation. In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176, Department of Electrical Engineering, Princeton University, 1970.

[22] D.S. Scott, Continuous lattices. In *Toposes, Algebraic Geometry and Logic (Lecture Notes in Mathematics, volume 274)*, pages 97–136, edited by F.W. Lawvere, Springer-Verlag, 1972.

[23] D.S. Scott, Lambda calculus: some models, some philosophy. In *The Kleene Symposium*, pages 223–265, edited by J. Barwise et al., North-Holland, 1980.

[24] D.S. Scott, λ-Calculus: Then & Now. Notes derived from Turing centennial conference presentation slides. [Online] Available: `http://turing100.acm.org/lambda_calculus_timeline.pdf` (2013).

[25] D.S. Scott and C. Strachey, *Toward a mathematical semantics for computer languages*, Oxford Programming Research Group Technical Monograph PRG-6, 1971.

Mark E. Hall
Department of Mathematics and Computer Science
Hastings College
Hastings, NE 68901 USA
Email: `mhall@hastings.edu`