SIAM J. COMPUT.
Vol. 5, No. 3, September 1976

THE RELATION BETWEEN COMPUTATIONAL AND DENOTATIONAL PROPERTIES FOR SCOTT'S D_∞-MODELS OF THE LAMBDA-CALCULUS*

CHRISTOPHER P. WADSWORTH†

Abstract. A prominent feature of the lattice-theoretic approach to the theory of computation due to D. Scott is the construction of solutions for isomorphic domain equations. One of the simplest of these is a domain isomorphic to the space of all continuous functions from itself to itself, providing the first "mathematical" model for the lambda-calculus of Church and Curry.

However, solutions of such domain equations are not unique; in particular, the lambda-calculus has many models. So the question arises as to which one should choose for computational purposes. We consider the relation between equivalence of meaning in Scott's models and the usual notions of conversion and reduction. By extending the lambda-calculus to allow approximate (i.e., partially specified) expressions and approximate reductions, we show that every expression determines a set of approximate normal forms of which it is the limit in Scott's model. Two immediate corollaries give a characterization of those expressions whose value is the least element of the model and further justification for the result that various lambda-calculus fixed-point combinators are all equal to the lattice-theoretic least fixed-point operator.

We show also that this leads to a characterization of equivalence which has a natural counterpart for other languages; specifically, expressions have the same meaning in Scott's model just when either can serve in place of the other in any "program" without altering its "global" properties.

Key words. lambda-calculus, lattices, isomorphic domain equations, projections, theory of computation, denotational semantics, equivalence, termination, head normal form, approximations, approximate normal form, incompleteness, fixed-point operators

Introduction. The purpose of this paper is to give an overview of recent results about the λ-calculus models discovered by Scott [17] in 1969, and to discuss how they reflect, and can be interpreted in terms of, the assumptions underlying the lattice-theoretic approach to the theory of computation. A few of the longer, more technical proofs have been omitted to make the development more readable and will be given in forthcoming papers [23], [24].

Ultimately our study concerns the acceptability of (some) language definitions based on the lattice-theoretic approach. This theory provides solutions for *isomorphic domain equations*, which are needed to define denotational semantics for many programming languages. However, solutions of such domain equations are not unique, and several different construction methods are now available. As a result, languages defined in this way may have many possible semantics and the question arises as to which of these is appropriate for reasoning about programs written in the language.

* Received by the editors March 13, 1975, and in revised form November 23, 1975.

† Department of Systems and Information Science, Syracuse University, Syracuse, New York. Now at Department of Computer Science, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland. This work was supported by National Science Foundation under Grant GJ-41540 and previously by a U.K. Science Research Council Grant to C. Strachey, Programming Research Group, Oxford University Computing Laboratory, Oxford, England.

To answer this question requires consideration of the relation between a proposed semantics for a language and the computational behavior of its programs. We present our study as an *example* of investigations of this kind. For our purposes, we can think of the λ-calculus as a programming language, albeit a restricted one, whose denotational semantics is provided by the lattice-theoretic models. The models we study are based on solutions of

$$\mathbf{D} \cong [\mathbf{D} \to \mathbf{D}] \tag{1}$$

where $[\mathbf{D} \to \mathbf{D}]$ denotes some suitable notion of function space from a domain $\mathbf{D}$ to itself. This is one of the simplest examples of an isomorphic domain equation and was historically the first for which a solution, called $\mathbf{D}_\infty$ by Scott, was found. We explore the relation between equivalence of meaning in these models and the usual notions of conversion and reduction, which can be regarded as (part of) a possible implementation.[1]

At the same time, the conversion rules provide a *proof theory* for the λ-calculus, so questions of completeness arise. We shall see that we do not have, and cannot expect, completeness in the usual sense. More precisely, two expressions are said to be interconvertible when they can both be reduced to a common expression (not necessarily irreducible); then although interconvertible expressions will be equivalent in all the possible semantics we are considering, for each possible semantics in general there will also be noninterconvertible expressions which are equivalent. For our analysis, the problem is that those examples which we would like to regard as equivalent are independent of any semantics whereas the set of noninterconvertible expressions having the same meaning varies as the semantics varies. From this point of view, the reasonableness of a semantics becomes the question of which semantics induces the "correct" equivalences between noninterconvertible expressions. We shall see that Scott's $\mathbf{D}_\infty$-model exhibits an interesting "limit completeness" property which leads to a characterization of equivalence of meaning in $\mathbf{D}_\infty$ having an obvious and natural counterpart for programming languages.

Our investigation is not the first example of its kind. Though the technicalities show few similarities, a good example for comparison is provided by studies of recursive function calculi, beginning with Kleene and more recently in work of, e.g., Cadiou [6] and Rosen [16]. In these calculi one is interested in definitions of functions by recursion, such as

$$f(n) = \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else } f(n-1) + 2n - 1. \tag{2}$$

Under a *semantic* explanation of recursion, (2) is regarded as an equation to be solved for f; as such it may have a unique solution, or many solutions, or no solutions at all. Definition (2) can also be explained *algorithmically* via a collection

[1] In practice, of course, reductions are often simulated by representing λ-expressions and other intermediate items of interest in various kinds of data structures—stacks, symbol tables, pointers, return links, etc.—so the idea of reduction is already an abstraction from "the real world". However, it may serve as a useful intermediary by deriving, independently of the models studied in this paper, relations between the results of evaluation-by-reduction and the computations of actual λ-calculus machines, several of which are described in the literature (e.g., in [10], where further references also may be found).

of calculation rules (the "copy-rule", simplification rules for conditionals, arithmetic, etc.) for transforming expressions; a set of ordered pairs $\langle n, m\rangle$ of integers may then be called a *computed function* of (2) if $f(\bar{n})$ can be transformed to $\bar{m}$ by some sequence of applications of the rules, where $\bar{n}$ and $\bar{m}$ denote numerals corresponding to the integers n and m, respectively. Under suitable restrictions on the language, it can be shown that equations such as (2) can always be solved uniformly—there is always a *least-defined* solution—and that the rules can be chosen and applied so that the algorithmically computed function agrees with this solution. This is well-known as the first recursion theorem of Kleene [9, § 66, Thm. 26]. The results in § 5 below provide a λ-calculus analogue of this result; indeed, when applied to the special case of several λ-calculus fixed-point operators they are essentially the same result (see also Morris [11]). Our study is thus exactly the same in nature as these earlier ones; the novel feature, as we have indicated, is that we consider here a language whose semantics requires solutions of domain equations.

After a quick review of the λ-calculus in § 1, we present its semantics in § 2 for arbitrary solutions, as complete lattices, of the isomorphism (1). In § 3 we extract the properties of Scott's particular solution, $\mathbf{D}_\infty$, we shall use and give some examples. Section 4 departs from the main study of models to introduce the equivalent notions of solvability and head normal form which play a vital role in interpreting the later results and understanding their proofs. In § 5 we generalize the ordinary notion of reduction to study "approximations" and develop their "limit" properties, which lead in § 6 to the mentioned characterization of equivalence of meaning in $\mathbf{D}_\infty$. Section 7 then discusses one of the more surprising properties of $\mathbf{D}_\infty$—the possibility of equivalences between normal forms and expressions with no normal form—and its implications.

Though much of the development is highly technical, nevertheless the results provide some new insights which should apply equally to more expressive and more realistic languages; some indications will be given in the conclusion.

1. The λ-calculus.

We assume familiary with the basic theory as presented, e.g., in [7], but give a brief summary to fix our notation and terminology.

We assume denumerably many *variables* $x, y, z, \cdots, x', y', \cdots$, and define the set of well-formed expressions, called *terms*, inductively as follows:

1. Every variable is a term.
2. If M and N are terms, so is the *combination* (MN), the parts M and N being called its *rator* and *rand*, respectively.
3. If x is a variable and M is a term, then the *abstraction* $(\lambda x.M)$ is a term, the parts x and M being called its *bv* and *body*, respectively.

We shall use $\equiv$ for syntactic identity of terms and, to reduce the proliferation of parentheses, we adopt the conventions that

(a) omitted parentheses in combinations associate to the left, e.g.,

$$xyzw \equiv (((xy)z)w),$$

(b) the scope of the dot " . " in an abstraction extends as far to the right as possible, i.e., to the first unmatched ") " or to the end of the term if that occurs first, and

(c) consecutive abstractions may be collapsed to a single one, e.g.,

$$\lambda xyz.M \equiv (\lambda x.\ (\lambda y.\ (\lambda z.M)))$$

An occurrence of a variable x in a term M is *free* if it is not inside the body M' of some part of M of the form $\lambda x.M'$, and *bound* otherwise. FV(M) denotes the set of variables occurring free in M. M is *closed* if FV(M) is empty, otherwise M is *open*; closed terms are sometimes also called *combinators.*

The notion of a *context* is the "dual" of the notion of a subterm and is useful for a uniform treatment of results for both open and closed terms. A context, $\mathbf{C}[\ ]$, consists of all parts of a term except that one subterm is missing (indicated by the empty brackets); $\mathbf{C}[M]$ then denotes the result of filling the missing subterm with M. Thus, the notation $\mathbf{C}[M]$ distinguishes a particular occurrence of M as a subterm. Note that a statement that terms A, B differ only in an occurrence of M, N, respectively, as a subterm, can then be expressed as the existence of a context $\mathbf{C}[\ ]$ such that $A \equiv \mathbf{C}[M]$ and $B \equiv \mathbf{C}[N]$. We shall see several such uses of contexts below when we compare the applicative properties of terms and their substitution instances. Many contexts we meet will be *head contexts*, of the general form

$$(\lambda x_1x_2 \cdots x_n.[\ ])M_1M_2 \cdots M_nA_1A_2 \cdots A_m, \quad n \geqq 0, \quad m \geqq 0.$$

In particular examples, $x_1, x_2, \cdots, x_n$ will typically be a list of free variables of the terms being considered and $M_1, \cdots, M_n, A_1, \cdots, A_m$ will be closed terms; the choice of $M_1, \cdots, M_n$ determines corresponding substitution instances, and the choice of $A_1, \cdots, A_m$ prescribes arguments for application of these substitution instances as functions.

We shall write $[N/x]M$ for the *substitution* of N for (free occurrences of) x in M. We omit a formal definition but assume it is given so that bound variables in M are changed when necessary to prevent capture of free variables of N; e.g.,

$$[yz/x](\lambda y.x(\lambda x.xy)) \equiv \lambda w.yz(\lambda x.xw),$$

where w is some variable different from x, y, and z. This ensures that substitution is well-defined for *all* terms M and N; see [7, pp. 89–104] for a full discussion.

Terms are "evaluated" by eliminating abstractions as much as possible, according to two replacement rules and an auxiliary rule allowing renaming of bound variables:

1. *α-conversion*: Provided $y \notin$ FV(M), a term of the form $\mathbf{C}[\lambda x.M]$ may be converted to $\mathbf{C}[\lambda y.M']$, where $M' \equiv [y/x]M$ and $\mathbf{C}[\ ]$ is any context.
2. *β-conversion*: A term of the form $R \equiv (\lambda x.M)N$ is called a *β-redex* and $R' \equiv [N/x]M$ is called its *contractum.* In any term, the operation of replacing an occurrence of R by R' is called a *β-contraction*; in the other direction, replacement of an occurrence of R' by R is called a *β-expansion* or *β-abstraction.* A sequence of (possibly zero) β-contractions is called a *β-reduction,* written $X\ \beta$-**red** X'; when β-abstractions may also be included in a sequence of replacements, the sequence is called a *β-conversion,* written $X\ \beta$-**cnv** X'.
3. *η-conversion*: A term of the form $\lambda x.Mx$, with $x \notin$ FV(M), is called an *η-redex,* and then M is its contractum. The definitions of *η-contraction,* etc., are analogous to those for β-contraction, etc., in 2.

When it is of no interest which rules are being applied, we write simply X **red** X' or X **cnv** X'.

A term is said to be *in normal form* if it does not contain a redex as a subterm. Terms in normal form are said to be *distinct* if they are not α-interconvertible. A term N in normal form is said to be a *normal form of* M iff M **cnv** N. (Again, we may prefix β- and/or η- to these notions as appropriate; when used without a prefix, we always mean β-η-normal form.) It is readily seen that every term in normal form can be written in the form

$$\lambda x_1 x_2 \cdots x_n . z N_1 N_2 \cdots N_m, \qquad n \geqq 0, \quad m \geqq 0,$$

with each N_i also in normal form; conversely, every such term is in β-normal form (and in β-η-normal form if also $N_m \not\equiv x_n$ or if x_n occurs free in $zN_1N_2 \cdots N_{m-1}$).

Not all terms have a normal form. The two best-known examples are $\Delta\Delta$, where $\Delta \equiv \lambda x.xx$, and the so-called paradoxical combinator

$$Y_\lambda \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

A third example is the term

$$J \equiv Y_\lambda(\lambda f.\lambda x.\lambda y.x(fy))$$

which we shall see behaves very much like the identity $I \equiv \lambda x.x$.

For our discussion of interpretations and models, we summarize the theory as a formal system. As such the λ-calculus is an equational calculus; there is one predicate symbol " = " and the formulae consist of all equations $M = N$ between terms. The axioms and rules of inference are the usual ones for equality, plus the three conversion rules:

A1. = is a *substitutive equivalence relation*:

(ρ) $\quad M = M$

(σ) $\quad \dfrac{M = N}{N = M}$

(τ) $\quad \dfrac{M = L,\ L = N}{M = N}$

(Subst) $\quad \dfrac{M = N}{\mathbf{C}[M] = \mathbf{C}[N]}$, for all contexts $\mathbf{C}[\ ]$.

A2. *conversion rules*:

(α) $\quad \lambda x.M = \lambda y.[y/x]M,$ provided $y \notin \mathrm{FV}(M)$,

(β) $\quad (\lambda x.M)N = [N/x]M,$

(η) $\quad \lambda x.Mx = M,$ provided $x \notin \mathrm{FV}(M)$.

Then, formally, M **cnv** N means that $M = N$ is provable from these axioms, and M **red** N means that $M = N$ is provable without the use of the symmetry rule (σ). Prefixes α-, β-, or η- on **cnv** or **red** denote probability without some of the conversion rules; e.g., M β-**red** N means that $M = N$ is provable without the use of (σ), (α) and (η).

Note that since $(\lambda x.Mx)X$ β-**red** MX, when $x \notin \mathrm{FV}(M)$, for all terms X, the η-rule is equivalent to the requirement of (functional) *extensionality*:

$$\text{(Ext)} \quad \frac{MX = NX \text{ for all } X}{M = N}$$

More essentially, however, the η-rule, when included, expresses a limitation on possible interpretations; it says that *all* terms can be regarded as functions (which, incidentally, we regard as equivalent if they have the same graph).

THEOREM 1.1 (The Church–Rosser Theorem). *If X* **cnv** *Y, there is a term Z such that X* **red** *Z and Y* **red** *Z. Hence, if a term has two normal forms X and Y, then X α-***cnv*** *Y.*

THEOREM 1.2 (The Standardization Theorem). *If M has a normal form, then M can always be reduced to normal form by* normal-order reduction, *defined as the reduction in which each step is determined by contraction of the* leftmost *redex* (i.e., the redex whose left-hand end is furthest to the left).

Two terms M and N will be said to be *separable* iff there is a (head) context $\mathbf{C}[\ ]$ such that $\mathbf{C}[M]$ **cnv** $I \equiv \lambda x.x$ and $\mathbf{C}[N]$ **cnv** $K \equiv \lambda x.\lambda y.x$.

THEOREM 1.3 (Böhm [5]). *If M and N have distinct β-η-normal forms, then M and N are separable.*

Theorems 1.1 and 1.2 are well-known; in particular, Theorem 1.1 establishes the *consistency* of the λ-calculus system based on A1, A2. (In the absence of a negation operation, a formal system for the λ-calculus is said to be inconsistent if *all* equations between terms are provable, for which it suffices that $I = K$ is provable, since $II(KA)B$ β-**red** A and $KI(KA)B$ β-**red** B for all terms A and B.)

Theorem 1.3 is, for distinct normal forms, a form of converse of the Church–Rosser Theorem. The latter shows that distinct normal forms cannot be proved equal by the conversion rules; Theorem 1.3 shows that if one were ever to postulate, as an extra axiom, the equality of two distinct normal forms, the resulting system would be inconsistent. So the truth of equations between terms having a normal form is completely resolved by the theory of conversion—if any two such terms with distinct normal forms have the same value in a model, then all terms have the same value.

2. Lattice models of the λ-calculus.

Because of the type-free style of application allowed in the λ-calculus, the domain of any interpretation must include (at least up to isomorphism) a significant portion of its own function space. In this section we consider interpretations based on arbitrary solutions of the isomorphism

$$\mathbf{D} \cong [\mathbf{D} \to \mathbf{D}] \tag{2.1}$$

with $\mathbf{D}$ a *complete lattice*[2] and $[\mathbf{D} \to \mathbf{D}]$ the lattice of *continuous* functions from $\mathbf{D}$ to $\mathbf{D}$ under the "pointwise" partial ordering. For a general orientation and the

[2] For those who prefer, directed complete partial orderings (partially ordered sets with a least element in which every directed subset has a least upper bound) may be used throughout without significantly affecting our development or results. However, we prefer to work with complete lattices for simplicity and ease of comparison to Scott's papers (though the more sophisticated notions associated with continuous lattices will not be used here).

definitions of monotonicity, directed set, continuous function, projections, etc., we refer the reader to the referenced papers of Scott. We shall use the symbols $\sqsubseteq$, $\bigsqcup$, $\bot$, and $\top$ to denote, respectively, the *partial ordering*, the *least upper bound* (l.u.b.) operation, the *least element*, and the *greatest element* of complete lattices.

Anticipating the notation of the particular solution to be studied later, we let $\mathbf{D}_\infty$ be *any* solution of (2.1), as a complete lattice, with more than one element, and we express the isomorphism by two (continuous) functions

$$\mathbf{D}_\infty \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} [\mathbf{D}_\infty \to \mathbf{D}_\infty], \tag{2.2}$$

$$\Psi(\Phi(x)) = x, \quad \text{for all } x \in \mathbf{D}_\infty, \tag{2.3}$$

$$\Phi(\Psi(f)) = f, \quad \text{for all } f \in [\mathbf{D}_\infty \to \mathbf{D}_\infty]. \tag{2.4}$$

Let **VAR**, **EXP** and **ENV** denote the sets of all variables, terms and environments, respectively. By an environment, ρ, we mean an association of values in $\mathbf{D}_\infty$ with all variables, so $\rho : \mathbf{VAR} \to \mathbf{D}_\infty$ and **ENV** is the set of all functions from **VAR** to $\mathbf{D}_\infty$. The interpretation of variables is extended to all terms by a mapping

$$\mathcal{V} : \mathbf{EXP} \to [\mathbf{ENV} \to \mathbf{D}_\infty].$$

We shall use the emphatic brackets $[\![$ and $]\!]$ to enclose terms and write $\mathcal{V}[\![M]\!](\rho)$ for the *value*, or *denotation*, of M relative to the environment ρ. $\mathcal{V}$ is defined by structural induction, with one clause for each case in the syntax of terms:

(S1) $\quad \mathcal{V}[\![x]\!](\rho) = \rho[\![x]\!]$,

(S2) $\quad \mathcal{V}[\![MN]\!](\rho) = \Phi(\mathcal{V}[\![M]\!](\rho))(\mathcal{V}[\![N]\!](\rho))$,

(S3) $\quad \mathcal{V}[\![\lambda x.M]\!](\rho) = \Psi(\lambda \mathbf{d} \in \mathbf{D}_\infty . \mathcal{V}[\![M]\!](\rho[\mathbf{d}/x]))$,

where $\rho[\mathbf{d}/x] \equiv \rho' \in \mathbf{ENV}$ is given by $\rho'[\![x]\!] = \mathbf{d}$ and $\rho'[\![x']\!] = \rho[\![x']\!]$ for $x' \not\equiv x$.

In (S1), the value of a variable x is ascertained by looking up its denotation in ρ, which, in view of the functional nature of ρ, is achieved by application of ρ to x. In (S2), the value of the rator M is interpreted as a function by application of Φ; this function is an element of $[\mathbf{D}_\infty \to \mathbf{D}_\infty]$, so can be meaningfully applied to the value of the rand N.

Clause (S3) is a little more involved. An abstraction $\lambda x.M$ is naturally interpreted as a function from $\mathbf{D}_\infty$ to $\mathbf{D}_\infty$. When applied to any argument $\mathbf{d} \in \mathbf{D}_\infty$, the result of this function is the value of the body M in the environment ρ' identical to ρ in all respects, except that the bound variable x is now associated with the argument $\mathbf{d}$. As $\mathbf{d}$ varies over $\mathbf{D}_\infty$, $\mathcal{V}[\![M]\!](\rho[\mathbf{d}/x])$ determines a continuous function, which is rendered as an element of $\mathbf{D}_\infty$ by the isomorphism Ψ. (That this function is continuous is a consequence of the continuity of the isomorphism pair Φ, Ψ and the fact that expressions fashioned out of variables and continuous functions by *typed* abstraction and *typed* application are continuous in all their free variables.)

It is important to notice the distinction between the language being defined and the notation used to define it (the meta-language). In particular, note the two different uses of the λ-notation in (S3); on the left is the symbol "λ" of the type-free λ-calculus, on the right we have the typed λ-notation used to write down an expression for the meaning of the type-free notation. The second usage is convenient but is not essential to a presentation of the definition of $\mathcal{V}$; it can be avoided at the cost of introducing a name for the function occurring as the argument of Ψ (i.e., by writing:

$$\text{(S3}')\qquad \mathcal{V}[\![\lambda x.M]\!](\rho)=\Psi(f),\quad \text{where } f(\mathbf{d})=\mathcal{V}[\![M]\!](\rho[\mathbf{d}/x])).$$

To establish that the above interpretation of terms gives a *model* for the λ-calculus, we must say when equations between terms hold in $\mathbf{D}_\infty$, and show that the axioms about equality and conversion are then satisfied. We define terms as being *semantically equivalent*, or *equal in* $\mathbf{D}_\infty$, when they have the same value for all associations of values with their free variables:

$$M=_{\mathbf{D}_\infty}N \quad\text{iff}\quad \mathcal{V}[\![M]\!](\rho)=\mathcal{V}[\![N]\!](\rho)\quad \text{for all } \rho\in\mathbf{ENV}.$$

That this gives a substitutive equivalence relation is immediate from the corresponding properties of equality of lattice elements. The validity of the conversion rules then follows from the properties (2.3) and (2.4) of the isomorphism, obvious results about $\mathcal{V}[\![M]\!](\rho)$ being independent of values in ρ for variables not occurring free in M, and a preliminary *substitution lemma*: *For all terms M and N, variables x, and environments ρ,*

$$\mathcal{V}[\![[N/x]M]\!](\rho)=\mathcal{V}[\![M]\!](\rho[\mathcal{V}[\![N]\!](\rho)/x]).$$

The latter asserts that extending environments models substitution correctly; the proof is a tedious but straightforward induction on the structure of M.

In outline we have proved

THEOREM 2.1. *The interpretation* (S1)–(S3) *and the relation* $=_{\mathbf{D}_\infty}$ *provide a model for the λ-calculus system based on α-β-η-conversion*

$$M\,\alpha\text{-}\beta\text{-}\eta\text{-}\mathbf{cnv}\,N\qquad \textit{implies}\qquad M=_{\mathbf{D}_\infty}N.$$

Since we shall always have in mind the fixed interpretation above, it is convenient now to simplify our notation by

(i) allowing the terms themselves to stand for their values in $\mathbf{D}_\infty$, and

(ii) identifying elements of $\mathbf{D}_\infty$ with their image under the isomorphism Φ, Ψ.

For closed terms the ambiguity in (i) is in any case not very great. Using the semantic clauses (S1)–(S3) to fully expand $\mathcal{V}[\![M]\!](\rho)$, we obtain an expression (of the meta-language) which is independent of ρ, e.g.,

$$\begin{aligned}\mathcal{V}[\![\lambda x.(\lambda y.yx)x]\!](\rho)&=\Psi(\lambda\mathbf{d}_0\in\mathbf{D}_\infty.\Phi(\Psi(\lambda\mathbf{d}_1\in\mathbf{D}_\infty.\Phi(\mathbf{d}_1)(\mathbf{d}_0)))(\mathbf{d}_0))\\&=\lambda\mathbf{d}_0\in\mathbf{D}_\infty.(\lambda\mathbf{d}_1\in\mathbf{D}_\infty.\mathbf{d}_1\mathbf{d}_0)\mathbf{d}_0\end{aligned}$$

by property (2.4) and the identification convention (ii) for Φ, Ψ. Apart from a change of bound variable names, this last expression differs from the original term only in the type indications for its bound variables. If M does contain free variables, the ambiguity is only slightly greater; a similar expansion then shows that $\mathcal{V}[\![M]\!](\rho)$ depends only on the values of these free variables in the environment ρ.

Thus the use of the terms themselves to denote their values is sufficiently precise if we say that

1. free variables are regarded as ranging over $\mathbf{D}_\infty$,
2. when a term is applied as a function we understand that its image under Φ is intended,
3. when we write an abstraction $\lambda x.M$ we understand that x is restricted to ranging over $\mathbf{D}_\infty$, and by the abstraction we really mean the image of the corresponding function under Ψ.

By virtue of Theorem 2.1, the conversion rules preserve values so we may use them to manipulate terms used in this way as expressions for elements of $\mathbf{D}_\infty$. Whenever we do need to be careful about distinguishing between terms and their values, or between elements of $\mathbf{D}_\infty$ and their image in $[\mathbf{D}_\infty \to \mathbf{D}_\infty]$, or for emphatic reasons, we can fall back on the more formal notation and write in all the appropriate $\mathcal{V}$, ρ, Φ, Ψ, etc., but most of the time we can tolerate the ambiguity.

In general the converse of the implication in Theorem 2.1 cannot hold for *arbitrary* solutions of the isomorphism (2.2), but for terms having a normal form there is a positive result. From the observations at the end of § 1, we know that distinct normal forms must have distinct values provided the model is nontrivial (not all terms have the same value). The latter is immediate from $I \neq_{\mathbf{D}_\infty} K$. (Otherwise, since $II(Kx)y$ β-**red** x and $KI(Kx)y$ β-**red** y, we would have $x = y$ for all $x, y \in \mathbf{D}_\infty$, contradicting the assumption that $\mathbf{D}_\infty$ contains more than one element.)

With $\mathbf{D}_\infty$ being a lattice rather than a set a slightly stronger result holds. Just as lattice equality determined an equivalence relation between terms, so $\sqsubseteq$ induces a *quasi-ordering*:

$$M \sqsubseteq N \quad \text{iff} \quad \mathcal{V}[\![M]\!](\rho) \sqsubseteq \mathcal{V}[\![N]\!](\rho) \quad \text{for all } \rho \in \mathbf{ENV}.$$

(As a relation between terms, $\sqsubseteq$ fails to be a partial ordering only in that $M \sqsubseteq N \sqsubseteq M$ implies that M and N have the same value but not that they are the same term.) The relation $\sqsubseteq$ is easily seen to be substitutive:

$$M \sqsubseteq N \quad \text{implies} \quad C[M] \sqsubseteq \mathbf{C}[N] \quad \text{for all contexts } \mathbf{C}[\ ].$$

Then, since the terms I and K are incomparable under $\sqsubseteq$ (otherwise, we would have the same contradiction as above), it follows that all pairs of separable terms are incomparable; in particular:

THEOREM 2.2. *If M and N have distinct β-η-normal forms, then M and N are incomparable under $\sqsubseteq$.*

For terms without normal forms the situation is not so straightforward. Their properties are dependent on the deeper structure of particular models, and Theorem 2.2 may fail if either M or N fails to have a normal form.

Everything we have said so far holds for *all* models of the λ-calculus (or at least for all extensional lattice models), and we cannot expect the conversion rules to be complete for an arbitrary model. At the same time it is clear that we would not want completeness in this sense anyway. At first one's intuition might have been that a "natural model" for the λ-calculus is one in which terms have the same value just when they are interconvertible. However, the rules of conversion are only the *minimum* one should expect of an equational theory of "type-free" functions and there are several respects in which they are deficient; for instance, they do not allow any inductive arguments. More immediately, we shall see examples of terms which are not interconvertible but which exhibit the same computational behavior. We would like to regard such terms as having the same meaning.

3. A particular model: The use of projections.

We turn our attention now to the particular models discovered by Scott in 1969. First we give a brief outline of the construction in order to extract the properties of $\mathbf{D}_\infty$ we need in a convenient form (the laws of projection and application below).

Let $\mathbf{D}_0$ be any complete lattice and, inductively, $\mathbf{D}_{n+1} = [\mathbf{D}_n \to \mathbf{D}_n]$. The key to the construction lies in making this hierarchy of function spaces *cumulative* by "embedding" the lower-type spaces in the higher-type ones. These embeddings are described formally by a sequence of *projection pairs*

$$\mathbf{D}_n \underset{\psi_n}{\overset{\phi_n}{\rightleftarrows}} \mathbf{D}_{n+1}, \quad n = 0, 1, 2, \cdots,$$

$$\phi_{n+1}(x) = \phi_n \circ x \circ \psi_n, \qquad x \in \mathbf{D}_{n+1},$$

$$\psi_{n+1}(x') = \psi_n \circ x' \circ \phi_n, \qquad x' \in \mathbf{D}_{n+2}.$$

Then the *inverse limit* of the $\mathbf{D}_n$'s, i.e.,

$$\mathbf{D}_\infty = \{\langle x_n \rangle_{n=0}^{\infty} : x_n = \psi_n(x_{n+1}), x_n \in \mathbf{D}_n\}$$

is a complete lattice (under the "componentwise" partial ordering) and gives an isomorphism

$$\mathbf{D}_\infty \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} [\mathbf{D}_\infty \to \mathbf{D}_\infty]. \tag{3.1}$$

There are two known "parameters" in this construction—the choice of the initial domain $\mathbf{D}_0$ and the choice of the initial projection pair ϕ_0, ψ_0—which can be varied and one still obtains a solution of (3.1). In this and later sections our results hold for *any* $\mathbf{D}_0$ with more than one element (we shall see in § 6 why, other than this, the choice of $\mathbf{D}_0$ doesn't matter) but with ϕ_0, ψ_0 *fixed* as Scott's original projection pair:

$$\phi_0(x) = \lambda y \in \mathbf{D}_0.x, \qquad x \in \mathbf{D}_0,$$

$$\psi_0(x') = x'(\perp_{\mathbf{D}_0}), \qquad x' \in \mathbf{D}_1.$$

In fact the construction does more than solve the isomorphism (3.1) for the solutions have an internal structure too. There are also projection pairs

$$\mathbf{D}_n \underset{\psi_{\infty n}}{\overset{\phi_{n\infty}}{\rightleftarrows}} \mathbf{D}_\infty$$

embedding each $\mathbf{D}_n$ as the subspace

$$\mathbf{D}_n^{(\infty)} = \{\phi_{n\infty}(x) : x \in \mathbf{D}_n\} \subseteq \mathbf{D}_\infty$$

and inducing projection functions

$$P_n \equiv \phi_{n\infty} \circ \psi_{\infty n} : \mathbf{D}_\infty \to \mathbf{D}_n^{(\infty)}.$$

We can now "forget" the finite-type spaces $\mathbf{D}_n$ used in the construction and work entirely within $\mathbf{D}_\infty$, with the subspaces $\mathbf{D}_n^{(\infty)}$ and the projections P_n. These projections satisfy various equations (e.g., see [20, p. 64] or [15, pp. 114–115]) and determine essentially all the structure of the resulting λ-calculus models. For this it is convenient to redefine the subscript notation to stand for the projections; we now write, for $x, y \in \mathbf{D}_\infty$ and $f \in [\mathbf{D}_\infty \to \mathbf{D}_\infty]$,

$$\begin{array}{lll} x(y) & \text{for} & \Phi(x)(y), \\ f & \text{for} & \Psi(f), \\ x_n & \text{for} & P_n(x), \end{array}$$

Transliterating the properties of the projections P_n into this notation gives the following:

Laws of projection.

(P1) $x_n \sqsubseteq x_m \sqsubseteq x \quad \text{for } n \geqq m,$

(P2) $\perp_n = \perp \quad (\perp \text{ now denotes } \perp_{\mathbf{D}_\infty}),$

(P3) $x = \bigsqcup_{n=0}^{\infty} x_n,$

(P4) $(x_n)_m = x_{\min(n,m)},$

(P5) (Additivity) $(\bigsqcup X)_n = \bigsqcup\{x_n : x \in X\} \quad \text{for } X \subseteq \mathbf{D}_\infty.$

Laws of application.

(P6) (Extensionality) $x(z) = y(z) \text{ for all } z \in \mathbf{D}_\infty \leftrightarrow x = y,$
$x(z) \sqsubseteq y(z) \text{ for all } z \in \mathbf{D}_\infty \leftrightarrow x \sqsubseteq y,$

(P7) $x(y) = \bigsqcup_{n=0}^{\infty} x_{n+1}(y_n),$

(P8) $\perp(y) = \perp,$

(P9) $x_0(y) = x_0 = x(\perp)_0,$

(P10) $x_{n+1}(y) = x(y_n)_n.$

(Where parentheses are omitted, application takes precedence over projection; e.g., in (P10), $x(y_n)_n$ when fully parenthesized is $(x(y_n))_n$.)

Three derived laws which follow from the above are

(P11) $x_{n+1}(y) = x_{n+1}(y)_m \quad \text{for all } m \geqq n,$

(P12) $x_{n+1}(y) = x_{n+1}(y_m) \quad \text{for all } m \geqq n,$

(P13) $x = y \leftrightarrow x_n = y_n \quad \text{for all } n \geqq 0.$

Informally, we can read applications of these projection functions as entailing a *loss of information*, in the following sense. We have

$$\mathbf{D}_0^{(\infty)} \subseteq \mathbf{D}_1^{(\infty)} \subseteq \cdots \subseteq \mathbf{D}_n^{(\infty)} \subseteq \cdots \subseteq \mathbf{D}_\infty$$

and, for each $x \in \mathbf{D}_\infty$, the projections $x_n \in \mathbf{D}_n^{(\infty)}$ form an increasing sequence

$$x_0 \sqsubseteq x_1 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots \sqsubseteq x$$

of *approximations* to x; moreover, since

$$x_n = \bigsqcup \{w : w \in \mathbf{D}_n^{(\infty)}, w \sqsubseteq x\}$$

follows easily from the above, we can think of x_n as the *best* approximation to x available in the subspace $\mathbf{D}_n^{(\infty)}$. Properties (P3) and (P13) then state that every element of $\mathbf{D}_\infty$ is determined by these "finite" approximations.

In a similar way, (P7) states that application of an element of $\mathbf{D}_\infty$ as a function is determined by the functional behavior of its projections. Note, however, that (P7) does *not* imply that the terms in the l.u.b.-expression on the right are the *best* approximations $x(y)_n$ to the result of the application on the left (in general we only have $x_{n+1}(y_n) \sqsubseteq x(y)_n$), but only that their *join* gives the correct result. (Note also that (P7) is not a definition of application but a *derived property* satisfied by the projection functions together with the Φ-part of the isomorphism; in the strict notation (P7) reads

$$\text{(P7}'\text{)} \qquad \Phi(x)(y) = \bigsqcup_{n=0}^{\infty} \Phi(P_{n+1}(x))(P_n(y)) \quad .)$$

Properties (P9)–(P12) tell us how the individual projections of elements behave as functions. Property (P9) states that 0th projections are constant functions on $\mathbf{D}_\infty$; this is a consequence of the choice of the constant-function embedding $\phi_0 : \mathbf{D}_0 \to \mathbf{D}_1$ given above. For $(n+1)$st projections, the definition of $\mathbf{D}_{n+1}^{(\infty)}$ as the embedding of $\mathbf{D}_{n+1} \equiv [\mathbf{D}_n \to \mathbf{D}_n]$ in $\mathbf{D}_\infty$ implies that elements of $\mathbf{D}_{n+1}^{(\infty)}$ are essentially functions from $\mathbf{D}_n^{(\infty)}$ to $\mathbf{D}_n^{(\infty)}$; that is, $\mathbf{D}_{n+1}^{(\infty)} \cong [\mathbf{D}_n^{(\infty)} \to \mathbf{D}_n^{(\infty)}]$. Thus, the application $x_{n+1}(y)$ will always give a result in the subspace $\mathbf{D}_n^{(\infty)}$ (property (P11) above) and is independent of information contained in its argument y which is not contained in the best approximation y_n to y in $\mathbf{D}_n^{(\infty)}$ (property (P12) above). Further, considering the expression $x(y_n)_n$ on the right of (P10) as an operation on y, x is being applied to the nth projection $y_n \in \mathbf{D}_n^{(\infty)}$ of the argument and the nth projection $x(y_n)_n \in \mathbf{D}_n^{(\infty)}$ of the result is being taken; but from what we just said, this is how the *best* approximation $x_{n+1} \in \mathbf{D}_{n+1}^{(\infty)} \cong [\mathbf{D}_n^{(\infty)} \to \mathbf{D}_n^{(\infty)}]$ acts, so the application $x_{n+1}(y)$ gives the same result (property (P10) above).

With these laws we can do many useful calculations about the values of terms in $\mathbf{D}_\infty$. Three typical examples are

THEOREM 3.1. (a) (Scott [20]) $\Delta\Delta = \bot$,
(b) (Park [13]) $Y_\lambda = Y$,
(c) $J =_{\mathbf{D}_\infty} I$,

where $Y : [\mathbf{D}_\infty \to \mathbf{D}_\infty] \to \mathbf{D}_\infty$ *defined by*

$$Y(f) = \bigsqcup_{n=0}^{\infty} f^n(\bot)$$

is the lattice-theoretic least fixed-point operator *and the terms* Δ, Y_λ, I *and* J *are as given in* § 1.

Note how use has been made of the ambiguity between terms and their values in stating the theorem. In (a) we really mean that $\mathcal{V}[\![\Delta\Delta]\!](\rho) = \bot$ for all ρ. For (b), from lattice theory we know that the function Y maps continuous functions to their least fixed-points. Y itself also is continuous, so, by virtue of the isomorphism, we can think of Y as an element of $\mathbf{D}_\infty$; that is, we can faithfully "identify" Y with the element $Y_\infty \in \mathbf{D}_\infty$ given by

$$Y_\infty \equiv \Psi(\lambda x \in \mathbf{D}_\infty. Y(\Phi(x))) = \Psi(Y \circ \Phi)$$

Then, properly stated, (b) reads: for all ρ, $\mathcal{V}[\![Y_\lambda]\!](\rho) = Y_\infty$.

Proof. We give the calculation for (b) as an illustration. Part (a) can be proved by a similar calculation, but notice it follows immediately from (b), since $\Delta\Delta$ **cnv** $Y_\lambda I$ and $\perp$ is clearly the least fixed-point of I.

For (b), since Y_λ is, by β-conversion, a fixed-point operator, we have $Y \sqsubseteq Y_\lambda$. For the converse, by extensionality it suffices to show that $Y_\lambda(f) \sqsubseteq Y(f)$ for arbitrary f. Let $X \equiv \lambda x.f(xx)$. Then it suffices to show that

(3.2) $$X(X_n) \sqsubseteq f^{n+2}(\perp) \quad \text{for all } n \geqq 0,$$

for then

$$Y_\lambda(f) \ \mathbf{cnv}\ X(X) = X\left(\bigsqcup_{n=0}^{\infty} X_n\right) = \bigsqcup_{n=0}^{\infty} X(X_n), \quad \text{by (P3) and continuity,}$$

$$\sqsubseteq \bigsqcup_{n=0}^{\infty} f^{n+2}(\perp) = Y(f).$$

We prove (3.2) by induction on n. For $n = 0$, we have

$$X(X_0) = f(X_0(X_0)) = f(X_0) \sqsubseteq f(f(\perp))$$

by definition of X, (P9), and monotonicity, since $X_0 = X(\perp)_0 \sqsubseteq X(\perp) = f(\perp(\perp)) = f(\perp)$. For $n \geqq 0$, we have

$$X(X_{n+1}) = f(X_{n+1}(X_{n+1})) = f(X_{n+1}(X_n)), \qquad \text{by definition of } X \text{ and (P12),}$$

$$\sqsubseteq f(X(X_n)) \sqsubseteq f(f^{n+2}(\perp)),$$

by $X_{n+1} \sqsubseteq X$ and induction hypothesis.

For part (c), let $F \equiv \lambda f.\lambda x.\lambda y.x(fy)$, so $J \equiv Y_\lambda F$. That I is a fixed-point of F, and hence $J \sqsubseteq I$, is immediate since FI β-η-**red** I. We prove the stronger result:

THEOREM 3.2. *I is the only fixed-point of F in $\mathbf{D}_\infty$.*

Proof. Let J be any fixed-point of F, so J satisfies the equation

(3.3) $$J(x)(y) = x(J(y)).$$

We prove inductively that $J_n = I_n$ for all $n \geqq 0$, for which, by extensionality, it suffices to prove that $J_n(x)(y) = I_n(x)(y)$ for arbitrary x, y.

For $n = 0$ we have, using (3.3), (P9), (P8) and (P2),

$$J_0(x)(y) = J(\perp)_0(y) = J(\perp)(\perp)_0 = \perp(J(\perp))_0 = \perp,$$

$$I_0(x)(y) = I(\perp)_0(y) = \perp_0(y) = \perp.$$

For $n = 1$ we have, using also (P10) with $n = 0$,

$$J_1(x)(y) = J(x_0)_0(y) = J(x_0)(\perp)_0 = x_0(J(\perp))_0 = x_0,$$

$$I_1(x)(y) = I(x_0)_0(y) = x_0(y) = x_0.$$

Now assume, inductively, that $J_{n+1} = I_{n+1}$ for some $n \geqq 0$. Then

$$\begin{aligned} J_{n+2}(x)(y) &= J(x_{n+1})_{n+1}(y) = J(x_{n+1})(y_n)_n \quad \text{by (P10) twice} \\ &= x_{n+1}(J(y_n))_n = x_{n+1}(J(y_n)_n) \quad \text{by (3.3), (P11, P12) with } m = n \\ &= x_{n+1}(y_n) \quad \text{since, using the induction hypothesis,} \\ &\qquad J(y_n)_n = J_{n+1}(y) = I_{n+1}(y) = I(y_n)_n = y_n \end{aligned}$$

$$I_{n+2}(x)(y) = I(x_{n+1})_{n+1}(y) = x_{n+1}(y) = x_{n+1}(y_n). \qquad \square$$

The second part of Theorem 3.1 extends to a whole sequence of λ-calculus fixed-point combinators, defined by

$$Y^{(0)} \equiv Y_\lambda, \qquad Y^{(i+1)} \equiv Y^{(i)}(G), \qquad i \geqq 0,$$

where $G \equiv \lambda y.\lambda f.f(yf)$. First, note the following intimate connection between G and arbitrary fixed-point operators:

LEMMA 3.3. (a) *In any extensional model of the λ-calculus, an element ∇ is a fixed-point operator iff ∇ is a fixed-point of G:*

$$\nabla f = f(\nabla f) \quad \textit{iff} \quad G\nabla = \nabla.$$

(b) *For the lattice-theoretic least fixed-point operator Y,*

$$G(Y) =_{\mathbf{D}_\infty} Y =_{\mathbf{D}_\infty} Y(G).$$

Proof. Part (a), and hence the first half of (b), is immediate from the definition of G by β-reduction and extensionality. For the second half of (b),

$$G^n(\perp)(f) = f^n(\perp), \qquad f \in \mathbf{D}_\infty, \qquad \text{for all } n \geqq 0,$$

is easily proved by induction on n. Hence, using the definition of $\sqcup$ on function spaces,

$$Y(G)(f) \equiv \left(\bigsqcup_{n=0}^{\infty} G^n(\perp)\right)(f) = \bigsqcup_{n=0}^{\infty} G^n(\perp)(f) = \bigsqcup_{n=0}^{\infty} f^n(\perp) \equiv Y(f)$$

from which the result follows by extensionality. $\square$

(Similar to Theorem 3.2, it might now be asked whether Y is the only fixed-point of G in $\mathbf{D}_\infty$, but this fails—there are other (continuous) fixed-point operators on $\mathbf{D}_\infty$ besides Y. However, it turns out that Y is the only one which is λ-definable, because Y_λ is a maximal term under the ordering $\sqsubseteq$. (A proof of the latter will be included in [24].) In other words, it can be shown that *every* λ-calculus term having the fixed-point property has the least fixed-point operator Y as its value in $\mathbf{D}_\infty$. For the particular terms $Y^{(0)}, Y^{(1)}, \cdots$, however, we need not bother here with this interesting, but technically quite difficult, generalization.)

COROLLARY 3.4. *For all $i \geqq 0$, $Y^{(i)} =_{\mathbf{D}_\infty} Y$.*

Proof. The case $i = 0$ is just Theorem 3.1(b). If, inductively, $Y^{(i)} =_{\mathbf{D}_\infty} Y$, then

$$Y^{(i+1)} \equiv Y^{(i)}(G) =_{\mathbf{D}_\infty} Y(G) =_{\mathbf{D}_\infty} Y \qquad \square$$

Historically, the equivalence of these fixed-point combinators in $\mathbf{D}_\infty$, discovered originally by Park [13], provided the first example of the incompleteness of the conversion rules for these models, for it was known previously [4, p. 195] that the first two, $Y^{(0)}$ and $Y^{(1)}$, are not interconvertible. The motivation for wishing to regard all these fixed-point combinators equivalent will be discussed at the end of § 5.

Part (c) of Theorem 3.1 is a more unexpected example of incompleteness, for it gives an example of a term without a normal form which is equivalent to one in normal form. We shall return to discuss this example further in § 7, where we shall see it is not so unreasonable as it seems at first sight. For now, we note that the phenomenon is not special to the particular normal form I (it would perhaps be even more surprising if it was!):

COROLLARY 3.5. *For any normal form N, there is a term X with no normal form such that $N =_{\mathbf{D}_\infty} X$.*

Proof. Choose an occurrence of a variable x in N which is not the rator of a combination. (Such an occurrence always exists, e.g., choose the rightmost occurrence of a variable in N.) Let X be the term obtained by replacing this occurrence of x by the term $J(x)$. Then $N =_{\mathbf{D}_\infty} X$, since $J(x) =_{\mathbf{D}_\infty} I(x)$ **cnv** x, and the replaced occurrence of x not being the rator of a combination is sufficient to guarantee that X does not have a normal form. □

4. Solvability and head normal forms. Classical studies of the λ-calculus generally emphasize the significance of normal forms and tend to divide the terms into two classes—those with normal forms, whose "values" are perfectly defined, and those with no normal form, which are regarded as "meaningless" or "undefined". Such a division is too "discrete" (indeed we shall find reason to doubt the semantic significance of the distinction at all). Instead, just as Scott argues for data objects in general, we must recognize varying degrees of definedness; the "value" of a term may be defined in some respects but not in others. More especially, we cannot consistently regard a term as undefined just because it fails to have a normal form; some can be used to give defined results in nontrivial ways, and in fact it would be inconsistent to identify *all* terms not having a normal form:

Example 1. Let A be any term without a normal form, and let $M \equiv \lambda x.xIA$ and $N \equiv \lambda x.xKA$. Both M and N fail to have a normal form, but MK **β-red** I and NK **β-red** K. Identifying M and N would therefore imply $I = K$, which would render the λ-calculus inconsistent.

The concepts of solvability and head normal form are the beginnings of our analysis of the semantic significance of terms. We think of them primarily as allowing distinctions to be drawn between terms without normal forms, but the notions apply equally to all terms. Solvability was first studied in connection with the λ-definability of partial recursive functions by Barendregt [2], [3], to which we refer the interested reader for a fuller discussion.

We consider first the closed terms and how they behave when applied as functions:

Example 2. Consider the terms

$$M_1 \equiv \Delta\Delta, \quad \text{where } \Delta \equiv \lambda x.xx,$$

$$M_2 \equiv \Delta T, \quad \text{where } T \equiv \lambda x.xxx,$$

$$M_3 \equiv \Delta D, \quad \text{where } D \equiv \lambda x.\lambda y.xx.$$

(In fact M_3 β-**red** $Y_\lambda K$.) These three terms, none of which have a normal form, are particularly "hereditarily undefined". No matter how they may be applied as functions their computations will continue indefinitely: for all $k \geqq 0$ and all terms $X_1, X_2, \cdots, X_k$, the applications $M_i X_1 X_2 \cdots X_k$ fail to have a normal form, for each of M_1, M_2, M_3 above.

On the other hand, not all terms without normal forms behave in this way as functions. For example, the terms of Example 1 do not, nor does the paradoxical combinator Y_λ or the term J given in § 1; e.g.,

$$Y_\lambda(KA) \quad \beta\text{-}\mathbf{red} \quad A, \quad \text{for any term } A,$$

$$J(KA)B \quad \beta\text{-}\mathbf{red} \quad A, \quad \text{for any terms } A, B.$$

We define a *closed* term M as being *solvable* iff there is an integer $k \geqq 0$ and terms $X_1, X_2, \cdots, X_k$ such that $MX_1X_2 \cdots X_k$ has a normal form.

Terms having a normal form are always solvable; indeed, more strongly:

LEMMA 4.1. *If M is a closed term having a normal form N and X is* any *term, there exist $k \geqq 0$ and terms $X_1, X_2, \cdots, X_k$ such that*

$$MX_1X_2 \cdots X_k \quad \beta\text{-}\mathbf{red} \quad X \tag{4.1}$$

Proof. We can write the normal form N in the form $N \equiv \lambda x_1 x_2 \cdots x_n . zN_1N_2 \cdots N_m$. Since M, and therefore N, is closed, we have $z \equiv x_i$ for some i. Choosing $k = n$, $X_i \equiv \lambda y_1 y_2 \cdots y_m . X$, and $X_j \equiv I$ for $j \neq i$, the result follows. □

The following corollary is then straightforward from the property IX **red** X, and indicates the reason for the terminology "solvability":

COROLLARY 4.2. *A closed term M is solvable iff the equation* (4.1) *with $X \equiv I$ can be solved for $X_1, X_2, \cdots, X_k$ (for some $k \geqq 0$) iff for* all *terms X the equation* (4.1) *can be solved.*

For *open* terms we must consider also their substitution instances. For example, $x(\Delta\Delta)$ can never yield a normal form by being applied as a function, but its substitution instance $[KS/x](x(\Delta\Delta))$ is solvable provided S is solvable. We therefore define an open term as being solvable iff there is a substitution of closed terms for its free variables such that the resulting (closed) term is solvable. Or, uniformly for all terms M, M is *solvable* iff there is a *head* context $\mathbf{C}[\ ]$ such that $\mathbf{C}[M]$ has a normal form. Corollary 4.2 then extends to open terms in the same way.

The concept of head normal form provides a syntactical characterization of the solvable terms. First:

Example 2 (cont.). If we inspect the reductions of M_1, M_2, M_3 we find:

1. M_1 reduces only to itself.
2. Every term to which M_2 reduces is of the form $((\lambda x.xxx)T)TT \cdots T$.
3. Every term to which M_3 reduces is, after α-conversion, of the form $\lambda x_1 x_2 \cdots x_n.((\lambda x.\lambda y.xx)D)$.

These terms have the general form: either they consist of a β-redex, or of a β-redex followed by a finite number of arguments, or of a finite number of abstractions on terms of the latter form. That is, they are of the form

$$\lambda x_1 x_2 \cdots x_n.((\lambda x.M)N)X_1X_2 \cdots X_m, \qquad n \geqq 0, \quad m \geqq 0. \tag{4.2}$$

A term of the form (4.2) will be said to be *not in head normal form* and the redex $(\lambda x.M)N$ is then called its *head redex*. The reduction of a term in which each step is determined by contraction of the head redex (when it exists) will be called *head reduction*.[3] □

On the other hand, the terms of Example 1 are not of the form (4.2), nor are the terms

$$\lambda f.f((\lambda x.f(xx))(\lambda x.f(xx))), \qquad \lambda x.\lambda y.x(Jy)$$

to which Y_λ and J, respectively, are reducible. None of these terms are in (or have) a normal form, but they are in *head normal form*, of the general form

$$(4.3) \qquad \lambda x_1 x_2 \cdots x_n.zX_1X_2 \cdots X_m, \qquad n \geq 0, \quad m \geq 0, \quad z \text{ a variable,}$$

where $X_1, X_2, \cdots, X_m$ are arbitrary terms. (We leave it to the reader to show that every term can be written uniquely in one of the forms (4.2) or (4.3) for suitable n, m, X_i, etc.)

In (4.3) the variable z is known as the *head variable* and the term X_i as its ith *main argument*. Two head normal forms will be called *similar* iff they have the same head variable (after α-conversion, if necessary, so that bound variables agree) and the *difference* between the number of main arguments and the number of initial bound variables (i.e., $m-n$ in (4.3)) is the same for both. (The use of the difference here is a technical point connected with η-conversion. Most similar head normal forms we meet will be *strongly similar*, i.e., will have the same number of main arguments and the same number of initial bound variables. Note that in any case similar head normal forms can always be converted to strongly similar ones, by applying η-abstractions to the one with fewer initial bound variables.)

Analogous to normal-order reduction of terms to normal form, it can be shown that a term has a head normal form iff its head reduction terminates, and all head normal forms of a term are similar (strongly similar if η-conversions are excluded). (In fact, note that a head redex, when it exists, is also the leftmost β-redex, so the head reduction of a term consists of some initial segment of its normal-order reduction, or the whole of the latter if the term does not have a head normal form.)

It is instructive to compare (4.3) with the structure of normal forms given in § 1. In a head normal form (4.3) the main arguments $X_1, X_2, \cdots, X_m$ can be arbitrary terms, whereas a normal form requires these inner subterms also to be in normal form. Thus, a head normal form is a kind of "weak" normal form, which we might say is in normal form "at the first level".

It is easy to see that terms having a head normal form are solvable, for the proof of Lemma 4.1 uses only the fact that N is in head normal form. The converse can also be proved by syntactic means (by analyzing head reductions) but we shall see an easier model-theoretic proof in the next section.

[3] This definition of head reduction should not be confused with that of Curry [8, pp. 32, 157]. For a head redex, Curry's definition requires $n=0$ above; here it is not appropriate to limit ourselves to this case, because the binding of free variables does not affect (un)solvability.

The upshot of this discussion is that only those terms without normal forms which are in fact unsolvable can be regarded as being "undefined" (or better now: "totally undefined"); by contrast, all other terms without normal forms are at least partially defined. Essentially the reason is that unsolvability is preserved by application and composition—if U is unsolvable, so are UX and $U \cdot X$ for all terms X—which we have seen is not true in general for the property of failing to have a normal form.

To say that (some) terms without normal forms have a value which is not undefined seems at first a little disturbing, for after all every reduction of any term without a normal form will fail to terminate. In fact it is our traditional conception of termination—the result of a program is defined if its execution terminates and undefined if not—which is at fault here, as nonterminating programs cannot always be regarded as being "totally undefined". Consider, for instance, a language with output statements, so that programs whether they terminate or not may be producing intermediate output, possibly even infinite output (e.g.,

```
let n := 0;
while true do begin output Prime(n);
                    n := n+1 end
```

producing a list of the prime numbers). In such latter cases we might say the program computes its (infinitary) result "bit-by-bit" (the result of the program being the "union" of the partial outputs); then only those nonterminating programs which produce no intermediate output should be thought of as being "totally undefined".

For understanding the interpretation of λ-calculus terms in $\mathbf{D}_\infty$ this analogy with languages which allow intermediate output is a helpful one to have in mind, in the following sense. Consider "evaluating" a term M by reduction. If we find that M has a head normal form (4.3), we can output the "first-level" information about its initial bound variables, its head variable, and the number of its main arguments; then proceed recursively to evaluate (in parallel) the main arguments $X_1, X_2, \cdots, X_m$. If this process stops in finite time the original term M has a normal form and this is the information which will have been output. The process may fail to terminate in two ways: Either at some stage it may find a (sub)term without a head normal form, in which case the relevant component of the result is "totally undefined" as no information about it will ever be output, or the process may recurse indefinitely producing infinite output. (The latter would happen, e.g., for Y_λ and J.) For either case the total output gives, intuitively, *all* the information that can be computed about the "value" of the whole term M. These ideas harmonize very nicely with the general intuitions underlying Scott's approach to the theory of computation and will be pursued further in the next section.

So far we have considered only what happens when terms (or their substitution instances) are applied as functions. If functions are applied *to them*, then, even for an unsolvable term U as argument, we can always obtain, say, a normal form as result by using a constant function. But such usages are trivial as the use of any term in place of U would give the same result; more generally, we shall see that unsolvable terms can never have a nontrivial effect on the outcome of a reduction (Corollary 5.5).

Thus the interpretation of unsolvable terms as being "least defined" is a good one. It will not be surprising therefore when we find that they are exactly the terms with value $\perp$ in the $\mathbf{D}_\infty$-model. It has been shown by Barendregt [2, p. 91] that it is consistent to identify all unsolvable terms (again, an easier proof will be seen below), and from a computational point of view it is desirable to do so as they behave so alike.

5. Approximate reduction.

5. Approximate reduction. Our discussion so far has established models only for the equational part of the theory of the λ-calculus. In consequence we know that the process of reduction preserves meanings, but there is no direct interpretation of the reduction-relation itself in these models. In this section we shall see that none is needed, for the relation can be treated metatheoretically; that is, its properties will be seen to be implicit in the internal structure of the models.

We shall take up the idea of "evaluation-by-reduction" in a little more general formulation than at the end of the previous section, so as not to be dependent on any one method of reduction. Based on suggestions originally due to Scott,[4] we shall extend the ordinary λ-calculus to allow *partial terms* and *partial*, or *approximate*, *reductions*, then we can investigate *limits* of better and better approximate reductions and tie these in with the notion of limit already present in the lattice-models. The method is similar to the usual arguments justifying the use of *least* fixed-points in work on the theory of computation (cf. the comments on recursive function calculi in the Introduction and the treatment of **while**-statements in [19] or recursive procedures in [1]). The results will show that although the theory of conversion is too weak for proof purposes—not all true equations between terms which hold in $\mathbf{D}_\infty$ can be proved—there is a limiting sense in which the reduction rules are complete for purposes of evaluation.

We shall base our limiting process on β-reduction. The ideas can also be applied to η-reduction but we do not need to do so here as the latter plays rather a subsidiary role in evaluation; in any case if we can prove anything about limits using fewer approximations, the results remain true when more approximations are included.

Suppose we start reducing a term M, using any method and order of reduction we care to choose. If M has a normal form and the reduction actually reaches one, everything is fine and we can take the normal form obtained as the "value" of M. If not, conventionally the possibility is not considered of attributing a value to a reduction which apparently fails to terminate; one simple concludes that the initial term M does not appear to have a normal form. However, we need not be so pessimistic, for there is something that can be said, based on the intermediate terms in the reduction:

Example 3. Suppose M has been reduced to a term M', say

$$M' \equiv \lambda x.\lambda y.y(x(\underline{(\lambda z.P)Q}))(xy)(\underline{(\lambda w.R)S})$$

[4] In conversation, October 1970.

where P, Q, R, S are terms. At this stage we cannot tell whether M has a normal form or not, because there are still at least the two underlined redexes in M'. However, from the form of M', we can say that if M', and hence M, is going to have a normal form, this must be of the form

$$\lambda x.\lambda y.y(x(?))(xy)(?)$$

because any further (β)-reduction of M' can affect only the parts where we have written "?". Now adjoin to the λ-calculus the special constant symbol Ω to stand for such "undetermined" parts of normal forms. Then we shall say

$$A' \equiv \lambda x.\lambda y.y(x(\Omega))(xy)(\Omega)$$

is a *partial* normal form of M—partial because it tells us something of the structure of the normal form of M (if such exists), but does not give complete information. The obvious interpretation of Ω in the lattice-models is as the least element $\perp$:

$$\text{(S4)} \qquad \mathcal{V}[\![\Omega]\!](\rho) = \perp.$$

Then $\Omega \sqsubseteq X$ for all terms X, from which the substitutivity property gives $A' \sqsubseteq M'$ $(=M)$. Thus, A' is certainly *one* approximation to M, and we shall now call it an *approximate normal form* of M. (Of course, it is possible that A' is actually equal to M in $\mathbf{D}_\infty$, if the subterms which have been replaced by Ω turn out to have value $\perp$, but in general this will not be the case.)

Clearly this idea of approximating any parts remaining to be evaluated (i.e., β-redexes) by Ω can be applied to all terms in all possible reductions of M. In this way we obtain a whole set of approximations, each giving some, in general incomplete, information about M. The obvious question is whether, passing to their values in $\mathbf{D}_\infty$, their join gives *all* the information "contained" in the value of the starting term M.

Notice that the answer is immediate for any term which has a (proper) normal form, for then its normal form is one of its approximate normal forms, and for some obvious terms with value $\perp$, e.g., for the term $\Delta\Delta$ which reduces only to itself so that Ω is its only approximate normal form. More informative is:

Example 4. All terms to which Y_λ reduces are of the form $\lambda f.f^n(XX)$, where $X \equiv \lambda x.f(xx)$ and $n \geqq 0$, so the approximate normal forms of Y_λ are $\{\lambda f.f^n(\Omega) : n \geqq 0\}$. Thus there is an exact parallel here with the terms in the usual l.u.b.-expression for the least fixed-point operator. (This of course was part of our motivation for studying approximate reduction for arbitrary terms.)

To formalize the above we first introduce some further terminology. Expressions of this λ-Ω-calculus will be called λ-Ω-*terms*, or simply terms when no confusion is likely. A term will be said to be *in approximate normal form* if it does not contain a β-redex, and in *proper* normal form if additionally it does not contain any occurrences of Ω. A term A will be called a *direct approximant* of a term M iff A is in approximate normal form and matches M except at occurrences of Ω in A, and will be called the *best* direct approximant of M if Ω occurs in A only at subterms corresponding to the (outermost) β-redexes in M. (In Example 3, A' was the best direct approximant of M'; other direct approximants are, e.g.,

$\lambda x.\lambda y.y(\Omega)(\Omega y)(\Omega)$ and $\lambda x.\lambda y.\Omega(xy)(\Omega)$.) A term A will be called an *approximate normal form of M* iff A is a direct approximant of a term to which M is reducible.

(In fact, these definitions are slightly wider than in our earlier discussion for they allow subterms larger than the outermost β-redexes to be replaced by Ω in forming direct approximants; but clearly every direct approximant is $\sqsubseteq$ the best one (hence the terminology), so use of the wider definition does not affect limits.)

From (S4) and the properties of $\perp$, it is immediate that two *Ω-conversion* rules are valid in $\mathbf{D}_\infty$:

$$(\Omega_1) \qquad \Omega X = \Omega,$$

$$(\Omega_2) \qquad \lambda x.\Omega = \Omega.$$

We leave it to the reader to show

LEMMA 5.1. *If M β-**red** M' and B, B' are the best direct approximants of M, M', respectively, then $B \sqsubseteq B'$.*

So the best approximants of successive terms in a reduction form an increasing sequence. Unfortunately for any particular reduction of a term M such a sequence may converge to a limit smaller than the value of M. (In particular, normal-order reduction is sometimes inadequate in this sense; consider $M \equiv \lambda x.x(\Delta\Delta)(R)$ where R is any β-redex with value different from $\perp$.)

Considering *all* reductions, however, remedies the deficiency. We shall write $\mathscr{A}(M)$ for the set of approximate normal forms of M. (It follows easily from Lemma 5.1 and the Church–Rosser Theorem that $\mathscr{A}(M)$ is always a directed set.) In the statement of the limit theorem which follows we write in the $\mathscr{V}$ and ρ to emphasize the distinction between syntactic and semantic concepts:

THEOREM 5.2. *For all terms M and environments ρ,*

$$\mathscr{V}[\![M]\!](\rho) = \bigsqcup \{\mathscr{V}[\![A]\!](\rho) : A \in \mathscr{A}(M)\}.$$

Proof. Unfortunately there is room only for a few hints of the proof here; the full proof will be given in [23]. The main step consists of the development of a formal *typed calculus* for carrying out the kind of calculations with projections we did informally in § 3, via the notion[5] of a *type-assignment* for terms appropriate to these models. The "types" are integers and a type-assignment consists of an association of (arbitrary) integers with *every* subterm of a term; the intended interpretation is that the corresponding projection of the value of the subterm is to be taken. The properties of projections, in particular (P3), and continuity then imply, by structural induction on M, that $\mathscr{V}[\![M]\!](\rho)$ is equal to the l.u.b. of (the values of) all such typed-terms representing type-assignments for M.

Corresponding to the properties (P9) and (P10) of projections, two forms of *typed β-reduction* can be defined and shown to preserve the values of terms with type-assignments. The "well-foundedness" of (P9) and (P10) (i.e., application of an $(n+1)$st projection always decreases the level of the projections involved by one, and application of a 0th projection allows the argument to be replaced by Ω without changing the result) can then be used to show that every typed-term T, representing a type-assignment for M, can be reduced to, and hence is equivalent

[5] Suggested to us by J. M. E. Hyland, personal communication, January 1972.

to, a typed-term T' in approximate normal form. The result then follows by showing that the untyped term corresponding to T' is always one of the approximate normal forms of M. □

With this theorem we can explain our remarks about the "limiting completeness" of the reduction rules. The rules are adequate for generating the (r.e.) set of terms to which M reduces. As these are generated, so more and more of the approximate normal forms of M are obtained. The theorem asserts that if we are prepared to generate enough of the terms to which M reduces, then we can calculate an expression whose value approximates the value of M as close as we like—intuitively, we can "compute" (canonical representations for) the values of terms in $\mathbf{D}_\infty$.

In another, related sense, the theorem can be regarded as asserting that every term has a "normal form"; it's just that this may be an infinite expression (as was hinted at toward the end of § 4). Several methods for specifying *infinite normal forms* have been suggested (e.g., see [12]); when done in the right way we should be able to say that terms have the same meaning (in $\mathbf{D}_\infty$) iff they have the same infinite normal form.

As one immediate consequence of Theorem 5.2, we obtain the following complete characterization of those terms which have value $\perp$ (for all environments) in these models:

COROLLARY 5.3. *The following three conditions are equivalent*:

(a) *M is unsolvable.*

(b) *M does not have a head normal form.*

(c) $M =_{\mathbf{D}_\infty} \Omega$.

Proof. We prove (c) implies (a) and (b) implies (c). That (a) implies (b), or equivalently that not-(b) implies not-(a), follows by a proof similar to that of Lemma 4.1.

For (c) implies (a), we give the proof for closed terms M. (The extension to terms with free variables—in which case, for solvability, one must consider closed substitution instances—is straightforward using the rule (Ω_2).) Suppose $M =_{\mathbf{D}_\infty} \Omega$ but M is solvable. Then there exist terms $X_1, X_2, \cdots, X_k$ $(k \geqq 0)$ such that $MX_1X_2 \cdots X_k$ **cnv** I. But then, by using (Ω_1), we have

$$I =_{\mathbf{D}_\infty} MX_1X_2 \cdots X_k =_{\mathbf{D}_\infty} \Omega X_1X_2 \cdots X_k =_{\mathbf{D}_\infty} \Omega,$$

which is a contradiction.

For (b) implies (c), suppose M does not have a head normal form, so every term M' to which M reduces is not in head normal form. For any term M' not in head normal form, all direct approximants of M' are of the form

$$A' \equiv \lambda x_1 x_2 \cdots x_n.\Omega X_1 X_2 \cdots X_m, \qquad n \geqq 0, \quad m \geqq 0. \tag{5.1}$$

But $A' =_{\mathbf{D}_\infty} \Omega$ by (Ω_1) and (Ω_2). Hence all approximate normal forms of M are equal to Ω in $\mathbf{D}_\infty$, so $M =_{\mathbf{D}_\infty} \Omega$ by Theorem 5.2. □

The last two results are pleasing *semantic* properties of the models of § 3. They are quite natural in themselves and desirable from our earlier discussion, but we can find additional technical support via several formal (syntactic) results relating terms and their approximate normal forms when used as parts of larger expressions. In particular we have:

THEOREM 5.4. *For all terms M and contexts* $\mathbf{C}[\ ]$,

(a) $\mathbf{C}[M]$ *has a normal form iff* $\mathbf{C}[A]$ *has the same normal form for some* $A \in \mathcal{A}(M)$.

(b) $\mathbf{C}[M]$ *has a head normal form iff* $\mathbf{C}[A]$ *has a similar head normal form for some* $A \in \mathcal{A}(M)$.

Proof. That the normal forms in (a) (head normal forms in (b)) will be the same (similar) follows from the separability of distinct normal forms (dissimilar head normal forms). (For dissimilar head normal forms, this is straightforward; for distinct normal forms, see Theorem 1.3.) Part (a) is then proved by arguments based on the relative lengths of the normal-order reductions of terms and their direct approximants; we leave the full proof to [23]. Part (b) can be proved by analogous arguments about the lengths of various head reductions, but there is an alternative proof using our results above. By Corollary 5.3, it suffices to show

$$\mathbf{C}[M] \neq_{\mathbf{D}_\infty} \Omega \quad \text{iff} \quad \mathbf{C}[A] \neq_{\mathbf{D}_\infty} \Omega \quad \text{for some } A \in \mathcal{A}(M).$$

But this follows easily from continuity and Theorem 5.2. □

COROLLARY 5.5. *Suppose U is unsolvable and* $\mathbf{C}[\ ]$ *is any context. Then* $\mathbf{C}[U]$ *has a normal form* (*a head normal form*) *iff* $\mathbf{C}[M]$ *has the same normal form* (*a similar head normal form*) *for all terms M.*

Proof. The "if" part is trivial. We give the proof of the "only if" part for normal forms. Suppose $\mathbf{C}[U]$ has a normal form N. By Theorem 5.4(a), there is an approximate normal form A' of U such that $\mathbf{C}[A']$ **red** N. Since U is unsolvable, A' must be of the form (5.1) in the proof of Corollary 5.3. Hence

$$\mathbf{C}[\lambda x_1 x_2 \cdots x_n.\Omega X_1 X_2 \cdots X_m] \quad \mathbf{red} \quad N.$$

It is easily shown that normal-order reductions to (proper) normal form are not affected by applications of Ω-conversion or replacements of Ω by arbitrary terms, from which the result follows. □

Intuitively, Corollary 5.5 shows why unsolvable terms should be regarded as being "least defined". Theorem 5.4 shows that the "interesting" computational behavior of terms is determined by their approximate normal forms, so Theorem 5.2 can be interpreted as showing that the values of terms in $\mathbf{D}_\infty$ "contain" just the information relevant to their use in computations.

In fact, Theorem 5.4 can be extended to apply also to the *approximate* normal forms of $\mathbf{C}[M]$. In the case of a function application FM (i.e., taking $\mathbf{C}[\ ] \equiv F[\ ]$ so $\mathbf{C}[M] \equiv FM$) where F is any term, this specializes to: *For all $A' \in \mathcal{A}(FM)$, there is an $A \in \mathcal{A}(M)$ such that $A' \in \mathcal{A}(FA)$.* Or, equivalently:

$$\mathcal{A}(FM) = \bigcup \{\mathcal{A}(FA) : A \in \mathcal{A}(M)\}.$$

In words: To obtain an approximate normal form of a function application FM requires only an approximate normal form of the argument M. This is just a λ-calculus (syntactic) analogue of the general considerations motivating the continuity of functions in Scott's theory of computation.

Finally we close this section by considering again the fixed-point combinators of § 3. For $i = 0$ and $i = 1$ (and it seems likely for all $i \geqq 0$) *all* terms to which $Y^{(i)}$ reduces are of the form

$$\lambda f.f^n(\lambda x_1 x_2 \cdots x_k.RX_1X_2 \cdots X_m), \qquad n, k, m \geqq 0,$$

where R is a β-redex and X_j is a term, for $j=1,2,\cdots,m$. (For $Y^{(0)}$, we have $k=m=0$ and $R\equiv XX$ where $X\equiv\lambda x.f(xx)$; see Example 4. For $Y^{(1)}$, see [4, p. 195].) So the approximate normal forms of $Y^{(i)}$ consist essentially (i.e., after use of (Ω_1) and/or (Ω_2) if $m\neq 0$ and/or $k\neq 0$) of

$$\{\lambda f.f^n(\Omega):n\geqq 0\}$$

Thus, $Y^{(0)}$ and $Y^{(1)}$ have the same set of approximate normal forms—which, in view of Theorem 5.4, is why we would like to regard them as equivalent—and these approximate normal forms correspond exactly to the terms in the l.u.b.-expression for the least fixed-point operator—which is why we would like them to have the least fixed-point operator as their meaning.

6. A characterization of $=_{\mathbf{D}_\infty}$. The results of § 5 provide motivation for several semantic properties of Scott's λ-calculus models—roughly, they show that terms are given the right meanings from a computational point of view. In this section we show that equivalence of these meanings is also natural, by exhibiting a direct connection between $=_{\mathbf{D}_\infty}$ (and the ordering $\sqsubseteq$) and a syntactic relation definable from the conversion rules.

From our discussion so far, two particular equivalence relations between terms are worth noting:

1. *approximate normal form equivalent*: $M \sim_{\mathrm{a}} N$

$$M\sim_{\mathrm{a}} N \equiv_{\mathrm{def}} \mathscr{A}(M)=\mathscr{A}(N)$$

2. *normal form equivalent* (first studied by Morris [11]): $M\sim_{\mathrm{n}} N$

$M\sim_{\mathrm{n}} N\equiv_{\mathrm{def}}$ for all contexts $\mathbf{C}[\ ]$, $\mathbf{C}[M]$ has a normal form iff $\mathbf{C}[N]$ has (the same) normal form.

Then Theorems 5.4 and 5.2 imply

$$M\sim_{\mathrm{a}} N \Rightarrow M\sim_{\mathrm{n}} N,$$

$$M\sim_{\mathrm{a}} N \Rightarrow M=_{\mathbf{D}_\infty} N,$$

Further, it can be shown that

$$M\sim_{\mathrm{n}} N \Rightarrow M=_{\mathbf{D}_\infty} N,$$

but the reverse implication does not hold (e.g., consider I and J with the null context $\mathbf{C}[\ ]\equiv[\ ]$)—essentially what happens is that the property of having a normal form is too strong. However, if we replace "normal form" by "head normal form" throughout, as suggested by § 4, then implications can be established in both directions. That is, if we define *head normal form equivalence*, $M\sim_{\mathrm{h}} N$, analogous to $M\sim_{\mathrm{n}} N$, then

$$M\sim_{\mathrm{h}} N \Leftrightarrow M=_{\mathbf{D}_\infty} N$$

From right to left this is an easy consequence of Corollary 5.3 and substitutivity. In the other direction the full proof is rather intricate but can be derived quite straightforwardly with the aid of a lemma we shall state without proof (Lemma 6.2 below).

First, it is convenient to introduce two alternative formulations and corresponding quasi-orderings. We define *M is solvably extended by N* as

$$M \lesssim_s N \equiv_{\text{def}} \text{ for all contexts } \mathbf{C}[\ ], \text{ if } \mathbf{C}[M] \text{ is solvable, so is } \mathbf{C}[N].$$

and write $M \sim_s N$ if each is solvably extended by the other. We define *M is semi-separable from N* by

$$M \not\lesssim N \equiv_{\text{def}} \text{ there is } a \text{ (head) context } \mathbf{C}[\ ] \text{ such that } \mathbf{C}[M] \textbf{ red } I \text{ but } \mathbf{C}[N] \text{ is unsolvable,}$$

and call two terms *semi-separable* if either is semi-separable from the other. (The latter is a weak version of "separable" defined in § 1; the prefix "semi-" is used in the same sense as its usage in connection with decision procedures in computability theory.)

LEMMA 6.1. *For all terms M and N,*

(a) $M \sim_h N \Leftrightarrow M \sim_s N$,

(b) $M \not\lesssim N \Leftrightarrow M \not\lesssim_s N$.

Proof. The proof is immediate from the definitions, by Corollaries 5.3, 4.2, respectively. □

LEMMA 6.2. *Suppose A is in approximate normal form and N is any term. Then*

$$A \not\sqsubseteq N \Rightarrow A \not\lesssim N.$$

Proof. The proof is achieved by a generalization of the construction used by Bohm [5] in the proof of Theorem 1.3 and will be given in [24]. □

Now we can establish the characterization of $=_{\mathbf{D}_\infty}$. In fact it is more convenient to do so for the inequality $\neq_{\mathbf{D}_\infty}$, along with a characterization of $\not\sqsubseteq$:

THEOREM 6.3.[6] *The following three conditions are equivalent*:

(a) *M and N are semi-separable* $(M \not\lesssim N)$,

(b) *M and N are not solvably equivalent* $(M \not\lesssim_s N)$,

(c) $M \neq_{\mathbf{D}_\infty} N$ $(M \not\sqsubseteq N)$.

Proof. By Lemma 6.1(b), it suffices to show (a) is equivalent to (c).

To show (c) implies (a), suppose $M \not\sqsubseteq N$. Then $A \not\sqsubseteq N$ for some $A \in \mathscr{A}(M)$, by Theorem 5.2 and the properties of l.u.b's. By Lemma 6.2, there is a context $\mathbf{C}[\ ]$ such that $\mathbf{C}[N]$ is unsolvable and $\mathbf{C}[A]$ **red** I, whence also $\mathbf{C}[M]$ **red** I by Theorem 5.4(a), which shows M is semi-separable from N.

To show (a) implies (c), suppose $M \not\lesssim N$ and let $\mathbf{C}[\ ]$ be a context such that $\mathbf{C}[M]$ **red** I and $\mathbf{C}[N] \equiv U$ is unsolvable. Then $M \sqsubseteq N$ would imply $I \sqsubseteq U$, contradicting U being unsolvable (because of Corollary 5.3). □

For closed terms, Theorem 6.3 specializes to:

THEOREM 6.4. *If M and N are both closed terms, the following three conditions are equivalent*:

(a) $M \not\sqsubseteq N$,

(b) *there is a closed term F such that FM* **red** *I and FN is unsolvable,*

(c) *there exist closed terms* $X_1, X_2, \cdots, X_k$ $(k \geqq 0)$ *such that*

$$MX_1X_2 \cdots X_k \textbf{ red } I$$

while $NX_1X_2 \cdots X_k$ *is unsolvable.*

[6] Essentially this same result has also been proved independently by J. M. E. Hyland.

Proof. For (c) implies (b), simply set $F \equiv \lambda z.zX_1X_2 \cdots X_k$. That (b) implies (a) follows from Theorem 6.3 since, when (b) holds, $\mathbf{C}[\ ] \equiv F[\ ]$ is then a semi-separating context. For (a) implies (c), suppose $M \not\sqsubseteq N$, so that M is semi-separable from N, by Theorem 6.3. Let

$$\mathbf{C}[\ ] \equiv (\lambda x_1x_2 \cdots x_n.[\ ])M_1M_2 \cdots M_nA_1A_2 \cdots A_m$$

be a head context such that $\mathbf{C}[M]$ **red** I and $\mathbf{C}[N] \equiv U$ is unsolvable. Since M is closed, we have

$$\mathbf{C}[M] \equiv (\lambda x_1x_2 \cdots x_n.M)M_1M_2 \cdots M_nA_1A_2 \cdots A_m \quad \beta\text{-}\mathbf{red} \quad MA_1A_2 \cdots A_m$$

and similarly for N. Hence (c) follows by setting $k = m$ and $X_i \equiv A_i$ for $i = 1, 2, \cdots, k$. □

We can understand the content of Theorem 6.4 further with reference to the lattice structure of $\mathbf{D}_\infty$. For distinct elements $a, b \in \mathbf{D}_\infty$, say $a \not\sqsubseteq b$ for definiteness, there are always continuous functions which (partially) distinguish between them: for arbitrary $d \neq \perp$,

(6.1) $a \not\sqsubseteq b \Rightarrow$ there is a continuous $f: \mathbf{D}_\infty \to \mathbf{D}_\infty$ such that $f(a) = d$, $f(b) = \perp$.

(E.g., the function

$$f(x) = \mathbf{if}\ x \not\sqsubseteq b\ \mathbf{then}\ d\ \mathbf{else} \perp$$

has this property and is continuous.) Now call an element of $\mathbf{D}_\infty$ *λ-definable* iff it is the value $\mathcal{V}[\![M]\!](\rho)$ of a *closed* term M (since M is closed, the choice of ρ doesn't matter), and suppose a and b are λ-definable elements, say

$$a = \mathcal{V}[\![M]\!](\rho), \quad b = \mathcal{V}[\![N]\!](\rho).$$

Since $M \not\sqsubseteq N$ iff $a \not\sqsubseteq b$ (by definition of $\not\sqsubseteq$), (6.1) implies that when $M \not\sqsubseteq N$ there is a continuous function f which distinguishes between the values of M and N, and by the isomorphism Φ, Ψ this function f can itself be considered an element of $\mathbf{D}_\infty$. Theorem 6.4 expresses the stronger property that, for distinct λ-definable elements, there is always a *λ-definable* function which distinguishes between them. A second consequence, for closed terms M and N, is that

$$M =_{\mathbf{D}_\infty} N \quad \text{iff} \quad MZ =_{\mathbf{D}_\infty} NZ \quad \text{for all } \textit{closed} \text{ terms } Z$$

(which incidentally implies we have models satisfying the ω-rule of Barendregt [2, p. 48], for the restriction that M and N be closed terms can be removed with the aid of Theorem 6.3). So the λ-definable subspace has pleasant "closure" properties and the term structure of the model (i.e., the ordering $\sqsubseteq$ and the equality $=_{\mathbf{D}_\infty}$ between terms) depends only on the λ-definable subspace of $\mathbf{D}_\infty$.

We can also see, with hindsight, why the construction of $\mathbf{D}_\infty$ yields essentially the same model for arbitrary choice of the initial domain $\mathbf{D}_0$ (but not for arbitrary choice of the initial projection pair ϕ_0, ψ_0—the characterization theorems above hold only with the ϕ_0, ψ_0 mentioned in § 3). It is not difficult to show that, of the elements of $\mathbf{D}_0$ (regarded as the subspace $\mathbf{D}_0^{(\infty)}$ of $\mathbf{D}_\infty$), $\perp$ is the only one which is λ-definable. (In fact, $\perp$ is the only λ-definable element of the union of all the "finite-type" spaces $\mathbf{D}_n^{(\infty)}$, $n = 0, 1, 2, \cdots$.) For the choice of $\mathbf{D}_0$ this means that, although $\mathbf{D}_0$ must include two distinct elements, $\perp$ and $\top$, for a nontrivial model, additional elements are redundant as they will not be λ-definable and hence cannot affect the term structure of the resulting model.

Further insight is provided by recalling the intuitive interpretation of partial orderings in Scott's theory, in terms of "information content". For domain elements a and b, $a \not\sqsubseteq b$ was intended to mean "a gives some information where b gives either no information or different information". The results of § 5 showed that (the values of) terms having different computational behavior have differences in their "information content"; Theorem 6.3 gives the converse, that when there is a difference in the "information content" of (the values of) terms in $\mathbf{D}_\infty$, this can be detected in their computational behavior. Roughly, when $M \not\sqsubseteq N$, their semi-separating context searches the terms for the information contained in M but not in N. If and when this is found, i.e., for M, the appropriate part of M is extracted and "standardized" to give the identity as result. In the corresponding part of N one finds no information (an unsolvable term) or different information (a term with head normal form dissimilar to that in the corresponding part of M); in either case the standardizing stage for M can be chosen so that when applied to this part of N the resulting term is unsolvable. (In fact, the omitted proof of Lemma 6.2 is just a formalization of this informal sketch; the tricky part involves showing that all the selections and manipulations can be done within the λ-calculus.)

A more technical consequence concerns the possibility of giving additional axioms for the λ-calculus in an attempt to axiomatize the model completely. Although neither the relation $=_{\mathbf{D}_\infty}$ not its negation $\neq_{\mathbf{D}_\infty}$ are recursively enumerable (in fact, $=_{\mathbf{D}_\infty}$ is Π_2^0-complete), effective relative axiomatizations are possible. Since unsolvable terms are equal in $\mathbf{D}_\infty$, consider the extended theory λ^* in which we postulate the equality of all unsolvable terms:

$$\text{(Uns)} \qquad \frac{M, N \text{ both unsolvable}}{M = N}$$

as an additional inference rule. It is easily shown that any equality of semi-separable terms is inconsistent with (Uns), whence it follows from Theorem 6.3 that $M \neq_{\mathbf{D}_\infty} N$ iff $M = N$ would render the theory λ^* inconsistent. This implies that the *inequality* $\neq_{\mathbf{D}_\infty}$ (and similarly the relation $\not\sqsubseteq$) can be effectively axiomatized *relative to* λ^*; roughly, one constructs a proof of $M \neq_{\mathbf{D}_\infty} N$ by tracing backwards a proof of inconsistency of $\lambda^* + M = N$ (e.g., a proof of $I = K$ in $\lambda^* + M = N$). Such a relative axiomatization is of some interest as (Uns) is the only noneffective rule in λ^*—in fact, (Uns) is equivalent to the halting problem (implied by results in [2]). In a certain sense, this is the best one can do as regards axiomatizing the inequality $\neq_{\mathbf{D}_\infty}$ (because $=_{\mathbf{D}_\infty}$ is a Π_2^0-relation).

It also follows that $=_{\mathbf{D}_\infty}$ is the *largest, consistent* model of the theory λ^*. This has the interesting interpretation: given that we wish to consider all unsolvable terms equivalent, terms have different values in $\mathbf{D}_\infty$ iff this is necessary for consistency. Or, equivalently, terms are equal in $\mathbf{D}_\infty$ iff there is no reason to distinguish between them, where we regard the property of terms having different computational behavior (in the "global" sense we have discussed) as a necessary and sufficient reason for distinguishing between them.

7. Rationalization for $I = J$.

The possibility of equivalences between normal forms and terms with no normal form is one of the more unexpected, almost pathological, properties of the models we have studied and deserves a separate discussion. At first, this possibility was surprising because we felt there was a close connection between the property of having a normal form and termination of programs. In this way we hoped to find a model-theoretic characterization of "normal form" and hence suggest a semantic analogue of termination. In § 4, we saw reason to doubt the analogy, so our original motivation here no longer applies.

However, this alone cannot be said to imply that we should necessarily expect or desire that $I = J$; to rationalize this, more is needed. Of course, the general consequences of the characterization in § 6 provide some motivation, but there are several more specific arguments also.

First, under a suitable formalization of "infinite normal form", the term J will have an infinite normal form—roughly, this will be

$$J = \lambda x_0.\lambda x_1.x_0(\lambda x_2.x_1(\lambda x_3.x_2(\lambda x_4.x_3(\cdots))))$$

—while $I \equiv \lambda x.x$ is a finite normal form. So $I = J$ is a λ-calculus analogue of, e.g., $3 = 2.999\cdots$, in the sense that it is an example of a finite expression which is equivalent to an infinite one, just as $3 = 2.999\cdots$ is an example of a real number which has both a finite and an infinite decimal notation.

A second insight concerns the operation $\mathbf{H}$ of (a single) η-expansion viewed as an operation within the λ-calculus. $\mathbf{H}$ can be expressed by the equation

$$\mathbf{H}(x) = \lambda y.x(y), \qquad y \not\equiv x,$$

and is, of course, an identity operation in any extensional model. Suppose now that instead of performing just one η-expansion, we consider an infinite η-expansion operator obtained by recursively applying η-expansion to the newly introduced variable y:

$$\mathbf{H}_\infty(x) = \lambda y.x(\mathbf{H}_\infty(y)), \qquad y \not\equiv x.$$

This equation is now just that which is satisfied by the term J, so J represents an infinite number of applications of an identity operation. Iterating an identity operation an *infinite* number of times will not always given an identity operation (often it will give "undefined") but this is the case for the particular iteration involved in J. Perhaps the best way of expressing this is that when we write a recursive definition we can always compute results step-by-step via the recursion, but sometimes a solution can also be found as a closed formula (non-recursive).

Clearly $\mathbf{H}_\infty$ can be generalized to other operators, e.g., to ones which perform any finite number of η-expansions at the top level before recursing on some or all of the newly introduced variables; all such examples will have the identity function as their value in $\mathbf{D}_\infty$. In a certain sense, this method and that of Corollary 3.5 exhaust all possibilities of equivalences between normal forms and terms with no normal form,[7] so all such examples are essentially just variations on the $I = J$ one.

[7] More precisely, we mean here that, given any term N in normal form, by applying infinite η-expansion operators to some or all of its subterms (not necessarily the same operator for each subterm) we can obtain many terms equivalent to N which have no normal form. Under a suitable formalization of "infinite η-expansion operator", we conjecture that every term not having a normal form but equivalent to N in $\mathbf{D}_\infty$ can be obtained in this way, up to α-β-conversion. However, we shall not pursue this formally here as the ideas are more clearly seen in the general discussion of "infinite conversions" to follow, but see also the treatment of infinite normal forms in Nakajima [12].

It is also interesting here to compare the characterization of $=_{\mathbf{D}_\infty}$ with the more naturally occurring equivalence $\sim_n$ defined in § 6. All the above examples will fail for $\sim_n$ (because no normal form can be normal form equivalent to a term without a normal form), but it can be shown that, again, they essentially exhaust all examples of terms for which $M =_{\mathbf{D}_\infty} N$ but $M \not\sim_n N$.

An alternative formulation allows comparison with other calculi. We can say that I and J are equal by "infinite conversion" or are "interconvertible in the limit"; we illustrate this view in Fig. 1, where we have applied suitably chosen η-expansions to I and α-β-reductions to J. As these conversions proceed, it can be seen that the part where the terms differ is pushed deeper and deeper inside the terms. In other words, although I and J are not finitely interconvertible, they can be transformed, by means of the conversion rules alone, to terms which match each other up to arbitrarily large, finite "depth".

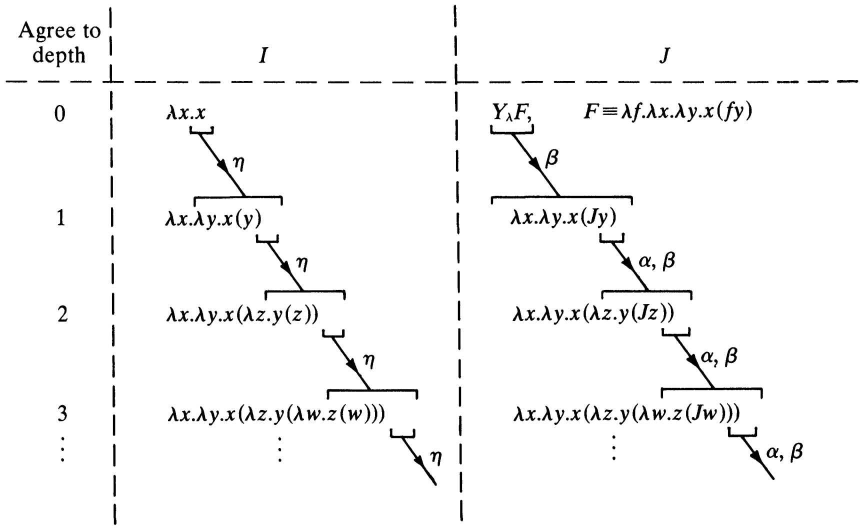

FIG. 1

This phenomenon can be generalized to give a full characterization of $=_{\mathbf{D}_\infty}$ along the same lines by taking account of the rule (Uns) equating unsolvable terms. ((Uns) was not needed in Fig. 1 as no subterm was unsolvable.) We have: *$M =_{\mathbf{D}_\infty} N$ iff M and N can be transformed by α-β-η-*Uns*-conversion to terms which agree to arbitrary depth.* More formally:

$$M =_{\mathbf{D}_\infty} N \quad \textit{iff} \quad (\forall k \geqq 0)(\exists M', N')(M' \simeq_k N' \quad \textit{and} \quad \lambda^* \vdash M = M', N = N')$$

where $\simeq_k$ symbolizes the notion of terms matching to depth k. By formalizing the notion of "depth" which we have indicated only rather loosely above, this formulation can be made the basis for an axiomatization of $=_{\mathbf{D}_\infty}$ with only two noneffective rules—the rule (Uns) and an infinitistic rule corresponding to the universal quantifier for k. (Again there is a similar axiomatization of the ordering $\sqsubseteq$).

By comparison, consider functions on the integers. One can give two definitions for the identity function, the explicit definition

$$f = \lambda n \in \mathbf{int}.n$$

and the recursive definition

$$g = \lambda n \in \mathbf{int}.\ \mathbf{if}\ n = 0\ \mathbf{then}\ 0\ \mathbf{else}\ g(n-1)+1$$

Although these two definitions clearly define the same function, one cannot give a recursive function calculus with a finite set of (effective) "conversion rules" such that the equivalence of f and g can be shown by transforming them into identical expressions. (Proofs of $f = g$ normally require the use of inductive methods.) However if we consider the rule

(R_k) Replace n by $\mathbf{if}\ n = k\ \mathbf{then}\ k\ \mathbf{else}\ n$

where n is an integer variable and k is a numeral, then, just as for I and J in Fig. 1, we can obtain "conversions" of f and g to expressions which agree to arbitrary depth. This is shown in Fig. 2, where, for g, we use the "copy-rule" to handle recursion plus the usual simplification rules of elementary arithmetic.

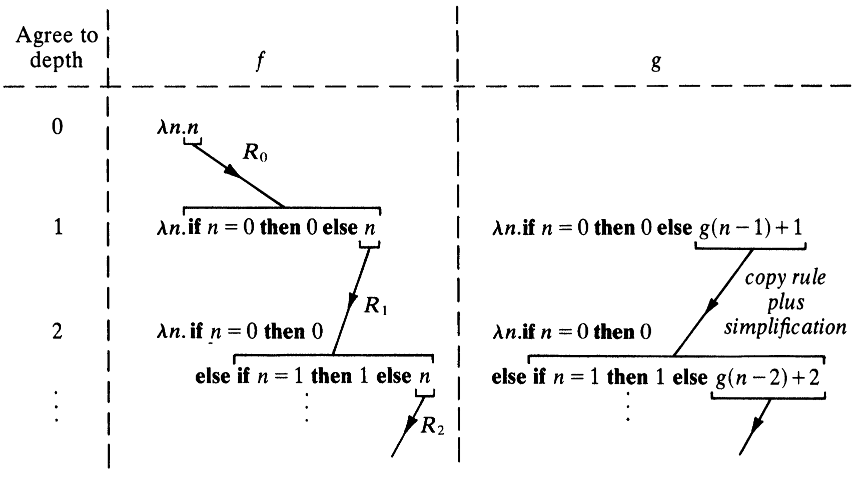
Agree to depth
f
g
0
1
2
λn.n
R0
λn.if n = 0 then 0 else n
R1
λn. if n = 0 then 0
else if n = 1 then 1 else n
R2
λn.if n = 0 then 0 else g(n − 1) + 1
copy rule plus simplification
λn.if n = 0 then 0
else if n = 1 then 1 else g(n − 2) + 2

FIG. 2

Taken together, we feel these arguments show that equivalences such as $I =_{\mathbf{D}_\infty} J$ are not unreasonable and, indeed, should be regarded as natural *whenever one accepts the validity of* η*-conversion.*

The proviso is, of course, crucial and at least inconvenient when dealing with most computer languages. We mentioned in § 1 that η-conversion embodies the view that *every* object is a function. There are no constants as such in the "pure" λ-calculus, one effect of which is that λ-calculus computations never quite touch ground level (e.g., by giving an integer, a truth value, or some other primitive object, as the result of a computation); if needed, constants must be represented as functions. Though this is not the most transparent approach for constants, it is a possible one and should have a corresponding "natural semantics". The properties of $\mathbf{D}_\infty$ can then be read as showing that when one is happy to represent everything as a function, the $\mathbf{D}_\infty$-model is a natural one, in the sense that the meanings of terms and their equivalences have the appropriate consequences for computational purposes.

A better approach, however, is to include primitive objects, or *atoms*, directly in the interpretation. Then one might study various extended λ-calculi which include constants in their syntax and some additional replacement rules (called δ-rules) for these. Models for such "applied" λ-calculi can be found via solutions of

$$\mathbf{E} \cong \mathbf{A} + [\mathbf{E} \to \mathbf{E}]$$

where $\mathbf{A}$ is some primitive lattice of atoms. The interpretation of terms runs much as before, with some given interpretation for the constants. (The only ambiguity concerns an application $a(x)$ in which a is an atom in $\mathbf{A}$; this is given a standard meaning, e.g., $\perp$, or $\top$, or a special *error*-atom.) Any solution for $\mathbf{E}$ will provide models for α-β-conversion but not, of course, for η-conversion (because distinct atoms behave the same as functions).

Particular solutions, $\mathbf{E}_\infty$, are known in which many of the results for $\mathbf{D}_\infty$ remain true, e.g., the analogues of Corollary 3.4 and Theorem 5.2. Whether all unsolvable terms will have value $\perp$ in $\mathbf{E}_\infty$ depends on the choice between "separated" and "coalesced" sums; that is, on whether one wishes to distinguish between the least element $\perp_{\mathbf{E}}$ and the least function $\perp_{[\mathbf{E}\to\mathbf{E}]}$. If so, this is mirrored in the syntactic aspects by calling an unsolvable term *strongly* unsolvable if also it is not reducible to an abstraction; then, with separated sums, terms will have value $\perp$ iff they are strongly unsolvable.

Equivalences such as $I = J$ will fail to hold. In fact, it can be shown that no normal form can be equal in $\mathbf{E}_\infty$ to a term without a normal form. This is to be expected as there is no reason to regard I and J as equivalent when η-conversion is denied; indeed, they can be distinguished by application to a constant.

The relation $=_{\mathbf{E}_\infty}$ will clearly be different from $=_{\mathbf{D}_\infty}$. One obvious conjecture is: $M =_{\mathbf{E}_\infty} N$ iff for all contexts $\mathbf{C}[\ ]$, $\mathbf{C}[M]$ has a normal form just when $\mathbf{C}[N]$ has the same normal form (here, normal form would mean β-δ-normal form). But with constants present, since whole programs are generally intended to give basic objects rather than functions as results, a more natural characterization would be

$$M =_{\mathbf{E}_\infty} N \Leftrightarrow \text{for all contexts } \mathbf{C}[\ ],\ \mathbf{C}[M] \text{ is reducible to a constant just when } \mathbf{C}[N] \text{ is reducible to the same constant.}$$

It seems likely that such a characterization can be established if sufficient constants (one for each atom?) and δ-rules are included in an extended calculus.

8. Conclusion. It is worth reflecting on how our results might generalize to other languages. Clearly there should always be some connection between the denotational semantics of a language and the properties of the computations which can be evoked by its programs. When such a connection has been established, we can then work with either the denotational or an implementation-oriented definition of the language. If we regard the denotational semantics as the primary definition, the connection would show that the implementation is not only correct but is "effectively complete", in the sense that it can compute and make use of every "bit" of information determined by the denotations. Alternatively, if we regard the implementation as given, such a connection establishes the denotational meanings as a valid basis for reasoning about the "global" properties of programs under the implementation (conceptually a simpler task than working with the implementation directly).

For proofs based on a denotational semantics, methods will be needed for performing inductions on the structure of domains, particularly domains obtained as solutions of isomorphic domain equations. As with the properties of λ-calculus models, many program properties will not be dependent on the internal structure of particular solutions, but it seems likely that some will be so dependent. In our study the laws of projections on $\mathbf{D}_\infty$ embody the relevant structure, but it remains to be seen whether a similar use of projections will provide a convenient framework for inductions on solutions of other domain equations.

The characterization of semantic equivalence of terms in $\mathbf{D}_\infty$ has an obvious counterpart for other languages in the form of a *substitutivity test*: program fragments may reasonably be regarded as equivalent just when either can serve in place of the other in any program without affecting the "input-output" behavior of the program. To emphasize the sense in which a program's behavior might be affected, we can generalize the notion of "semi-separable" as follows. Let π, π' be (well-formed) program fragments of the same syntactic category of a language $\mathbf{L}$. Call π and π' *semi-separable* iff there are programs Π and Π' of $\mathbf{L}$, differing only in an occurrence of π and π', respectively, such that the execution of Π halts giving some standard output (e.g., 0) while that of Π' would continue indefinitely giving no intermediate output, or vice versa. Then, given a semantics for $\mathbf{L}$, one can conjecture that π and π' are semantically distinct just when they are semi-separable. If this holds we can reasonably claim that the given semantics for $\mathbf{L}$ is the right one. (Of course, with the benefit of hindsight, these suggestions are not surprising.)

Finally, note the closure properties of the definable subspace of $\mathbf{D}_\infty$. In general, any solution of a domain equation will contain nondefinable elements and it would be distressing if two definable elements were equivalent in all the ways they may be combined with definable elements yet could be distinguished by making use of the nondefinable elements; for in the semantics based on such a domain there would be programs which have different meanings yet have the same computational behavior. (Of course, the results of § 6 show this does not occur for the λ-calculus semantics based on $\mathbf{D}_\infty$.)

Such a situation is not necessarily irretrievable, however. One possibility is simply that a language may be lacking in generality or expressive power; this could be patched either by extending the language directly or by regarding computational equivalence as being determined with respect to an extended language. A second possibility is that the definition of semantic equivalence could be modified; instead of defining semantic equivalence with reference to *all* environments, one can consider only environments in which the denotation of every variable is a definable element of the domains used in the semantics (with similar modifications for equivalence of functional abstractions). However, the structure of the domains may not then be as directly applicable to equivalence proofs as it was for the λ-calculus examples in § 3. A recent paper of Plotkin [14] points out these and other difficulties and expands on the possibilities.

Acknowledgments. I wish to thank Professors Dana Scott, Christopher Strachey, and John Reynolds for many helpful suggestions and encouragement. I am especially indebted to Martin Hyland for the notion of type-assignment used in the proof of Theorem 5.2.

REFERENCES

[1] J. W. DE BAKKER AND W. P. DE ROEVER, *A calculus for recursive program schemes*, Proc. IRIA Colloquium on Automata, North-Holland, Amsterdam, 1972.
[2] H. P. BARENDREGT, *Some extensional term models for combinatory logics and λ-calculi*, Ph.D. thesis, Utrecht Univ., the Netherlands, 1971.
[3] ———, *Solvability in λ-calculi*, Proc. 1972 Orléans Congrès de Logique, J. P. Calais, J. Derrick and G. Sabbagh, eds., to appear.
[4] C. BÖHM, *The CUCH as a formal and descriptive language*, Formal Language Description Languages, T. B. Steel, ed., North-Holland, Amsterdam, 1966, pp. 179–197.
[5] ———, *Alcune proprieta delle forme β-η-normali nel λ-K-calcolo*, Consiglio Nazionalle delle Recerche, no. 696, Rome, 1968.
[6] J. M. CADIOU, *Recursive definitions of partial functions and their computations*, Ph.D. thesis, Stanford Univ., Stanford, Calif., 1972.
[7] H. B. CURRY AND R. FEYS, *Combinatory Logic*, vol. 1, North-Holland, Amsterdam, 1958.
[8] H. B. CURRY, J. R. HINDLEY AND J. SELDIN, *Combinatory Logic*, vol. 2, North-Holland, Amsterdam, 1972.
[9] S. C. KLEENE, *Introduction to Metamathematics*, Van Nostrand, New York, 1952.
[10] C. MCGOWAN, *Correctness results for lambda-calculus interpreters*, Ph.D. thesis, Cornell Univ., Ithaca, N.Y., 1971.
[11] J. H. MORRIS, *Lambda-calculus models of programming languages*, Ph.D. thesis, Project MAC, Mass. Inst. of Tech., Cambridge, Mass., 1968.
[12] R. NAKAJIMA, *Infinite norms for λ-calculus*, Symposium on λ-calculus and Computer Science Theory, Consiglio Nazionalle delle Ricerche, Rome, 1975.
[13] D. M. R. PARK, *The Y-combinator in Scott's lambda-calculus models*, Symposium on Theory of Programming, Univ. of Warwick, unpublished, 1970.
[14] G. D. PLOTKIN, *LCF considered as a programming language*, J. Theoret. Computer Sci., to appear.
[15] J. C. REYNOLDS, *Notes on a lattice-theoretic approach to the theory of computation*, Dept. of Systems and Information Science, Syracuse Univ., Syracuse, N.Y., 1972.
[16] B. K. ROSEN, *Tree-manipulating systems and Church–Rosser theorems*, J. Assoc. Comput. Mach., 20 (1973), pp. 160–187.
[17] D. SCOTT, *Lattice-theoretic models for the λ-calculus*, Princeton Univ., Princeton, N.J., unpublished, 1969.
[18] ———, *Outline of a mathematical theory of computation*, Proc. 4th Ann. Princeton Conf. on Information Sciences and Systems, Princeton Univ., Princeton, N.J., 1970, pp. 169–176.
[19] ———, *The lattice of flow diagrams*, Semantics of Algorithmic Languages, E. Engeler, ed., Springer Lecture Notes in Mathematics, no. 188, Springer-Verlag, New York, 1971, pp. 311–366.
[20] ———, *Data types as lattices*, Notes, Amsterdam, unpublished, 1972.
[21] ———, *Lattice theory, data types and semantics*, Formal Semantics of Programming Languages, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 65–106.
[22] ———, *Lattice-theoretic models for various type-free calculi*, Proc. 4th Internat. Congr. for Logic, Methodology, and the Philosophy of Science (Bucharest), North-Holland, Amsterdam, 1973.

[23] C. P. WADSWORTH, *Approximate reduction and lambda-calculus models*, this Journal, to appear.

[24] ———, *A general form of a theorem of Böhm and its application to Scott's models for the λ-calculus*, in preparation.